PERSONAL GRAPHICS
PERSONAL GRAPHICS
PERSONAL GRAPHICS
PERSONAL GRAPHICS
PERSONAL GRAPHICS
PERSONAL GRAPHICS
PERSONAL GRAPHICS

PERSONAL GRAPHICS

FOR PROFIT AND PLEASURE ON THE **APPLE®** **II PLUS** COMPUTER

MICHAEL P. BARNETT · GRAHAM K. BARNETT

LITTLE, BROWN MICROCOMPUTER BOOKSHELF

050NPT 164
PRICE 14.50
M 36M 9000

PERSONAL GRAPHICS

Little, Brown Computer Systems Series

Gerald M. Weinberg, *Editor*

Barnett, Michael P., and Graham K. Barnett
Personal Graphics for Profit and Pleasure on the Apple® II Plus Computer

Basso, David T., and Ronald D. Schwartz
Programming with FORTRAN/WATFOR/WATFIV

Chattergy, Rahul, and Udo W. Pooch
Top-down, Modular Programming in FORTRAN with WATFIV

Coats, R. B., and A. Parkin
Computer Models in the Social Sciences

Conway, Richard, and David Gries
An Introduction to Programming: A Structured Approach Using PL/I and PL/C, Third Edition

Conway, Richard, and David Gries
Primer on Structured Programming: Using PL/I, PL/C, and PL/CT

Conway, Richard, David Gries, and E. Carl Zimmerman
A Primer on Pascal, Second Edition

Cripps, Martin
An Introduction to Computer Hardware

Easley, Grady M.
Primer for Small Systems Management

Finkenaur, Robert G.
COBOL for Students: A Programming Primer

Freedman, Daniel P., and Gerald M. Weinberg
Handbook of Walkthroughs, Inspections, and Technical Reviews: Evaluating Programs, Projects, and Products, Third Edition

Graybeal, Wayne, and Udo W. Pooch
Simulation: Principles and Methods

Greenfield, S. E.
The Architecture of Microcomputers

Greenwood, Frank
Profitable Small Business Computing

Healy, Martin, and David Hebditch
The Microcomputer in On-Line Systems: Small Computers in Terminal-Based Systems and Distributed Processing Networks

Lemone, Karen A., and Martin E. Kaliski
Assembly Language Programming for the VAX-11

Lias, Edward J.
Future Mind: The Microcomputer—New Medium, New Mental Environment

Lines, M. Vardell, and Boeing Computer Services Company
Minicomputer Systems

Mashaw, B. J.
Programming Byte by Byte: Structured FORTRAN 77

Mills, Harlan D.
Software Productivity

Monro, Donald M.
Basic BASIC: An Introduction to Programming

Morrill, Harriet
Mini and Micro BASIC: Introducing APPLESOFT, MICROSOFT, and BASIC PLUS

Mosteller, William S.
Systems Programmer's Problem Solver

Nahigian, J. Victor, and William S. Hodges
Computer Games for Businesses, Schools, and Homes

Nahigian, J. Victor, and William S. Hodges
Computer Games for Business, School, and Home for TRS-80 Level II BASIC

Orwig, Gary W., and William S. Hodges
The Computer Tutor: Learning Activities for Homes and Schools

Parikh, Girish
Techniques of Program and System Maintenance

Parkin, Andrew
Data Processing Management

Parkin, Andrew
Systems Analysis

Pizer, Stephen M., with Victor L. Wallace
To Compute Numerically: Concepts and Strategies

Pooch, Udo W., William H. Greene, and Gary G. Moss
Telecommunications and Networking

Reingold, Edward M., and Wilfred J. Hansen
Data Structures

Savitch, Walter J.
Abstract Machines and Grammars

Shneiderman, Ben
Software Psychology: Human Factors in Computer and Information Systems

Simpson, Tom, and Shaffer & Shaffer Applied Research & Development, Inc.
VisiCalc® Programming: No Experience Necessary

Walker, Henry M.
Problems for Computer Solutions Using FORTRAN

Walker, Henry M.
Problems for Computer Solutions Using BASIC

Weinberg, Gerald M.
Rethinking Systems Analysis and Design

Weinberg, Gerald M.
Understanding the Professional Programmer

Weinberg, Gerald M., Stephen E. Wright, Richard Kauffman, and Martin A. Goetz
High Level COBOL Programming

Windeknecht, Thomas G.
6502 Systems Programming

PERSONAL GRAPHICS

for profit and pleasure

on the Apple® II Plus Computer

MICHAEL P. BARNETT
Brooklyn College of the City University of New York

GRAHAM K. BARNETT
Princeton Day School

Little, Brown and Company
Boston Toronto

Library of Congress Cataloging in Publication Data

Barnett, Michael P.
Personal graphics for profit and pleasure on the Apple II Plus computer.

(Little, Brown computer systems series)
Includes index.
1. Computer graphics. 2. Apple II (Computer)
I. Barnett, Graham K. II. Title. III. Series.
T385.B36 1982 001.55'3 82-13000
ISBN 0-316-08164-7

Copyright © 1983 by Michael P. Barnett and Graham K. Barnett

All rights reserved. No part of this book may be reproduced in any form or by any electronic or mechanical means including information storage and retrieval systems without permission in writing from the publisher, except by a reviewer who may quote brief passages in a review.

Library of Congress Catalog Card No. 82-13000

ISBN 0-316-08164-7

9 8 7 6 5 4 3

MV

Published simultaneously in Canada
by Little, Brown & Company (Canada) Limited

Printed in the United States of America

Apple and Apple II Plus are registered trade marks of Apple Computer Inc.

Disclaimer of Liabilities: Due care has been exercised in the preparation of this book to insure its effectiveness. The authors and publisher make no warranty, expressed or implied, with respect to the programs or other contents of this book. In no event will the authors or publisher be liable for direct, indirect, incidental, or consequential damages in connection with or arising from the furnishing, performance, or use of this book.

Foreword

As Series Editor for the Little, Brown Computer Systems Series, I have the opportunity to read about sixty manuscripts each year. As you might imagine, not all of them prove fascinating, so I was delighted last summer to open an envelope and find a manuscript entitled *Personal Graphics for Profit and Pleasure.*

I was just then contemplating the purchase of a personal computer, and the question of how to achieve good graphics was high on my list. What I had been able to find on the subject until then fell into two categories:

1. Cookbooks full of PEEKs and POKEs that could be used to produce particular effects on particular hardware, but which were dead ends as far as generating anything of my own choosing.
2. Esoteric mathematical treatises that would enable me to generate masterpieces of graphic animation—after only three years of full-time study.

Personal Graphics, I was to discover, was in a category all on its own:

3. An easy-to-read, step-by-step guide to achieving simple graphic effects on my own computer, with minimal effort, that shows the way to grow into ever more complex effects, according to my own inclinations.

I knew I wanted this unique book in the Little, Brown Computer Systems Series, because I knew there were thousands of potential readers who shared my need. I was especially pleased to learn that the innovative method of typesetting would speed the process by which the book reached this eager audience.

But as I worked through *Personal Graphics*, I became aware of an even more important reason for making this book widely available. Not only was it unique in content and format, but it was also unique in method of development, for *Personal Graphics* is

4. A joyful record of discovery, shared by the author and his son while transforming their personal computer into an adventure in learning.

Anyone who is lucky enough to be a parent, or a teacher, ought to study *Personal Graphics* and learn how new technology can bring generations together. And anyone who's lucky enough to have an Apple II Plus—or, really, any personal computer—ought to work through *Personal Graphics* with a child, just for the fun and profit of it. And, if you don't have an Apple II Plus, or a child. perhaps you can borrow one!

Gerald M. Weinberg

Preface

This book is the product of two generations of Barnetts and several generations of computers. In selecting the topics that are addressed here, Graham, a student at Princeton Day School, and I have been influenced and helped by our complementary environments, insights, and interests. We hope that this book will provide a common experience across the generation gap for others. Personal graphics has burgeoned in the months since we started to write. There are many packages and languages that transcend the material we present. Some, such as LOGO and VISIPLOT, have major communities of users already. Nonetheless, a "free-standing" book, with programs that you can type and modify and use as prototypes, seemed worthwhile and in harmony with this growing mass of other material.

The production of the book, incidentally, has provided a practical test of a new development in electronic publishing that may have far-reaching consequences. The typesetting process was extended to produce the illustrations in place within the text, in the output of a Videocomp electronic typesetting machine. The data to create the illustrations were prepared on our Apple computer. Word processing and electronic typesetting technologies have made it possible for authors to produce their own publications directly. Now these can be illustrated inexpensively as well.

This book was typeset at Sedgwick Printout Systems using the PAGE-2 composition language that I designed in 1965 and techniques for computerized pictorial typesetting that I have developed over the past four years. Barbara, my wife, shared the work of producing the book.

I thought of the possibility of inexpensive pictorial typesetting in the course of my work on the production of instructional material by the use of computers, under the support of a grant from the Comprehensive Assistance to Undergraduate Science Education program of the National Science Foundation. I would like to thank Richard L. West of the N.S.F. for his encouragement and help in that project.

The detailed tactics used to produce this book have changed radically from those I experimented with initially, and I am indebted to many of the staff of Sedgwick for their help—in particular Edward R. Cotton, President, and Stanley R. Weiss, Manager, Proprietary Systems.

James M. Breslin of Computer Encounter has been a constant source of help and information about the Apple computer. George McLelland of Southeastern Software helped us establish an electronic dialog between the Apple and UNIVAC computers.

For their comments on many of the programs, and on their needs with respect to instruction in personal graphics, I am grateful to the seeming multitude of students that it has been my pleasure to teach at Brooklyn College in the Core course Introduction to Computer Science and Mathematical Reasoning and in my course on computer graphics, and to my colleagues on several campuses of City University in the Faculty Development Program that I taught.

Graham and I are grateful to Douglas O. McClure, until recently Headmaster of Princeton Day School, for providing an environment which led a student to write a book, and to Evan J. Romer at the school for introducing us to the Apple.

We would like to thank Charles F. Durang, now with Reston Publishers, for his role in the early development of the book, and Kenneth J. Tennity and Wayne P. Ellis of Little, Brown and Company for their help with editorial and production matters.

The programs in this book run on an Apple II Plus computer with 48K memory, one disk drive, and DOS. The programs can be purchased on disk from TYCON Corporation, CN 5258, Princeton, N.J., 08540.

Contents

1 What Is in This Book for You 1

2 Drawing Some Friends and Foes 7

2.1 Drawing a Face with Low-Resolution Graphics 7
2.2 A Little Winking and Blinking—Elementary Animation 10
2.3 Going to the Dogs 12
2.4 Sir Gruntgroan and High-Resolution Graphics 15
2.5 Centipedes and Chorus Lines 19
2.6 Pacing Things Out 22

3 Building Blocks and Shape Tables 27

3.1 A backdrop for *South Pacific* 27
3.2 Panoramas for Posterity 31
3.3 Building the Building Blocks 36

3.4 *South Pacific* Revisited 39
3.5 Using Apple Shape Tables 46
3.6 Snowflakes and Diatoms 50
3.7 Some Happy Squares 52
3.8 Throwing the Hammer 54

4 Going Around in Circles 57

4.1 How to Draw Circles 57
4.2 A Slight Case of *Jaws* 61
4.3 Jazzing It Up 64
4.4 Squaring and Denting a Circle 66
4.5 Kidneys, Hearts and Snails 68

5 A Little Plotting and Scheming 75

5.1 Graphics for Management 75
5.2 Creating a Font File 76
5.3 A Simple Graphing Program 85
5.4 Bars and Bar Charts 95
5.5 A Slice of Apple Pie 103

6 A Medley of Motley Curves 111

6.1 On the Path of a Flying Saucer 111
6.2 Curves That Go Up and Down 115
6.3 Making a Polynomial Fit 130
6.4 Some Curves to Smooth Out Angles 137
6.5 Doing the Actual Smoothing 140
6.6 Going into Orbit 145
6.7 A Little Roulette 148
6.8 How to Make Waves 151

7 Some Further Directions 163

7.1 Adding a Little Perspective 163
7.2 A Rotating Donut 167
7.3 A Kaleidoscope of Fun 170
7.4 More Simple Geometry 171

8 A Short Account of Applesoft 173

8.1 Languages for the Apple 173
8.2 Applesoft Programs 174
8.3 Initialization and Assignment 174
8.4 Numeric and String Expressions 175
8.5 Flow of Control 176
8.6 Input/Output Instructions 177
8.7 Graphics Commands 179
8.8 Using the Disk 181
8.9 System and Editing Commands 182

About this Book 185

Subject Index 189

Keyword Index 191

List of Programs

WINKFACE 11
DOGADD 14
ROBOT 18
CENTIPEDE 20
DRAW1 24
ALOHA1 30
ALOHA2 34
DESIGN1 38
DESIGN2 43
STATIC1 50
STATIC2 52
SQUARES 54
HAMMER 55
CIRCLES 60
MR. MOUTH 64
SON OF MR. MOUTH 65
CIRCLOIDS 67
ENVELOPES 73
FONT FORMER 80
PLOT1 88
PLOT2 89
PLOT3 91
BAR CHART 100
PIE CHART 108
GALILEO 114
POLYPLOT 125
POLYFIT 133
BEZFUN 140
BEZFIT 143
ORBIT 147
ROULETTE 150
SINEWAVE 158
ZVIEW 166
TRYTOR 169
KALEID 170

Chapter 1

What is in this book for you

Wherever we turn, Apple computers are at hand. Offices, homes, and classrooms seem to breed them. You can display pictures and graphs and eye-catching patterns on their screens in infinite variety. These increase enormously their potential to aid and to amuse you. At a very low cost, personal computers give hundreds of thousands of young and old the key to computer graphics. For years the expense of equipment kept this field of inestimable importance to a relative few. Now you can join the newly enfranchised and be at the forefront.

Many people want to catch the attention of their management or customers or students with facts and questions on a computer screen. You can do this by making letters dance, and by using pictures that are worth a thousand words. This book shows you how. It contains dozens of BASIC programs to get you started

- Using Apple II graphics in low and high-resolution
- Drawing straight line pictures on the screen
- Doing your own animation

- Plotting graphs and pie and bar charts
- Using PEEKs and POKEs, text files and shape tables, and other details of Apple software of vast use in graphics and other programs
- Drawing interesting patterns and creating exciting visual effects
- Producing smooth curves by sophisticated methods of computer-aided design
- Creating text in different fonts and sizes as part of a graphics display
- Drawing three-dimensional scenes in perspective
- Getting a measure of graphics packages you might want to buy

You do not have to fight any mathematics first. The programs capture this for you, and can be adapted to hosts of work and play situations. When any formulas are used, you do not need to know them. They are explained, however, to give you a handle on the mathematics if you want it. Each chapter suggests dozens of ways you can vary and expand the programs for profit and/or fun. These help

- Financial and other data analysis and display
- Making your own games
- Planning business and other institutional activities
- Teaching math and science and other subjects for many levels and ages
- Designing industrial and artistic products
- Challenging people who like to solve puzzles

Meet Cuthbert the Calculating Cur and Cyrus the Sensitive Centipede. See curves that look like snails and hearts and kidneys. Pluck up courage and look at a few equations—they don't bite, and no one will know if you dance around shouting "Eureka" as a result. If you know a little math already, take a look at the equation for a straight line that makes graphics programming easier but was almost lost for fifty years. See how simple some things like Bezier fitting really are. Whatever your interests, you may find some tips and tactics on programming your Apple.

And whatever your interests, whatever your age (in years or in spirit), read on, run the programs, profit, and enjoy. Watching an apple led Newton to the law of gravity. What can your Apple lead you to, even if you don't climb a tree to pick it?

For a start, Figures 1.1–1.10 are sketches of Apple computers with different attachments. The equipment shown in Figure 1.10 was used to draw Wilfred the Wishful, Figure 1.11. You can also connect your Apple to high-quality pen plotters, videodisk units, speech synthesizers, laboratory and industrial equipment for monitoring and control, and a seemingly limitless variety of other devices for special applications.

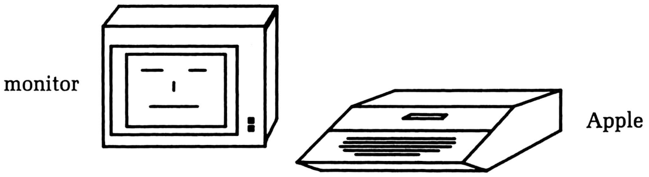


Figure 1.1. Apple computer with monitor. You can get black-and-white monitors, and also color monitors, in several sizes, or you can use a TV set instead.

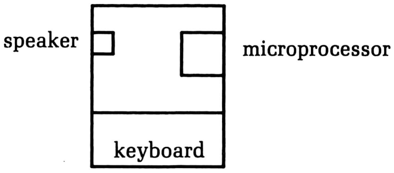

speaker
microprocessor
keyboard

Figure 1.2. Floor plan of an Apple computer. This shows the speaker that is used to make many different sounds.

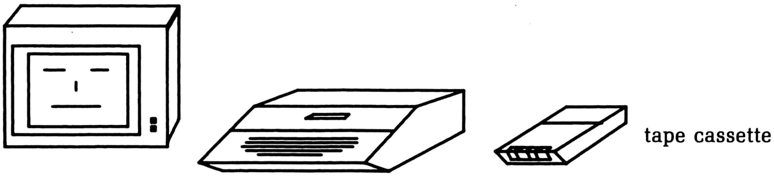

tape cassette

Figure 1.3. Apple, monitor, and tape cassette. You can store programs on the tape and rerun them. Also, you can use programs and data that other people have put on tapes for distribution.

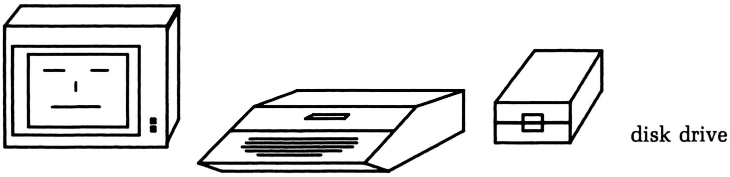


Figure 1.4. Apple, monitor, and disk drive. Floppy disks are more convenient than tapes for many applications.

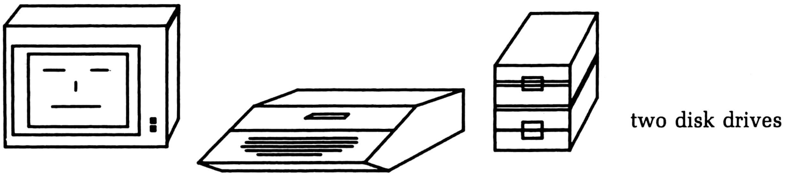

two disk drives

Figure 1.5. Apple, monitor, and two disk drives. Two drives make it easier to copy disks and to run programs that are very large or use bulky data.

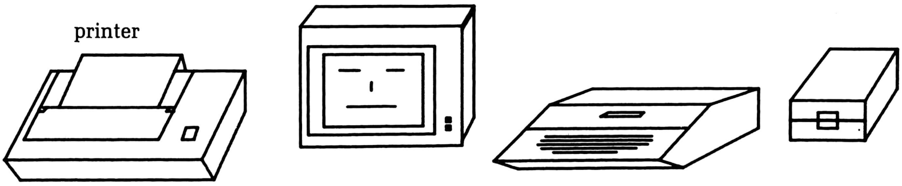


Figure 1.6. Apple, monitor, disk drive, and printer. You can get printers for text, or for text and graphics. You can print text in different fonts and sizes.

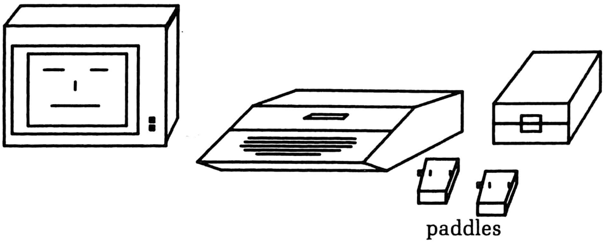


Figure 1.7. Apple with two paddles. You move the dial on a paddle to the setting that you want, and then press a button on the paddle to send this to the computer.

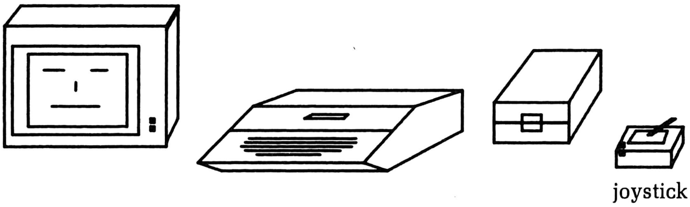


Figure 1.8. Apple with joystick control. You move the joystick to set a position anywhere on the screen.

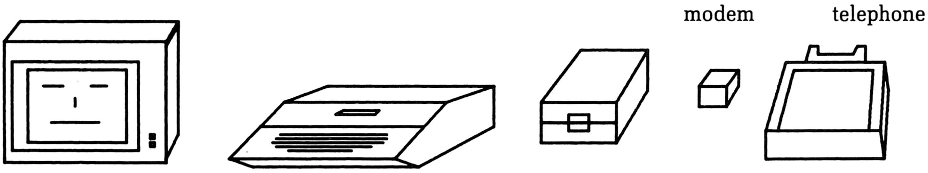


Figure 1.9. Apple equipped for communications. With a communications interface, modem and telephone, one Apple can talk back and forth with other personal computers, and with large computers too.

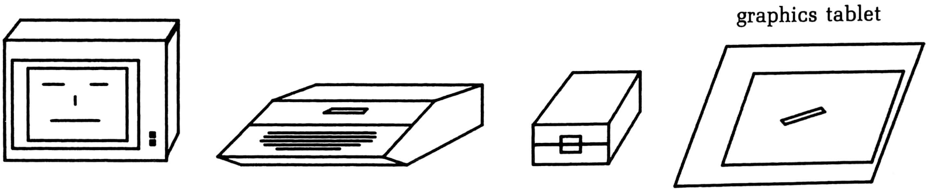


Figure 1.10 Apple with graphics tablet. You can draw pictures on the tablet for the computer to display on the monitor, store on disk, print, and transmit to other computers.

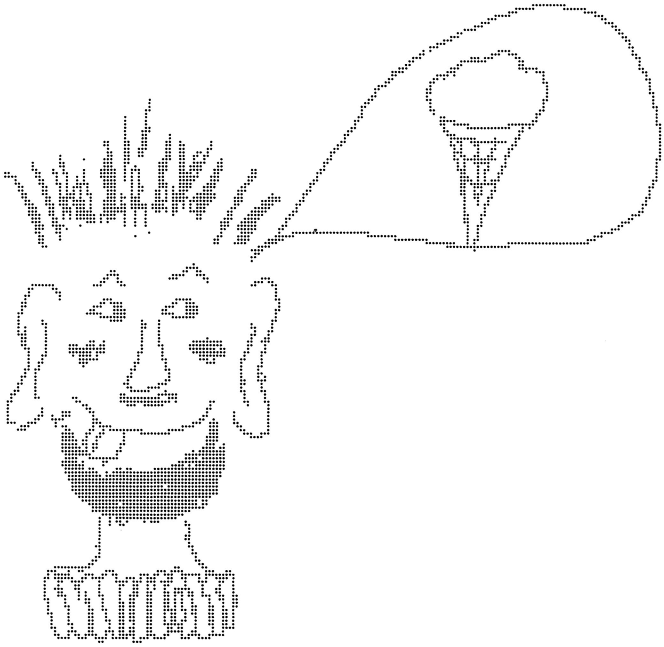


Figure 1.11. Wilfred the Wishful

A computer or a computer terminal can be connected to a telephone line via a "modem." This is a device that turns computer data into sounds which a telephone can transmit, and vice versa. By connecting your Apple to the telephone via a modem, you can let your Apple "talk" to other Apples around the world, transmitting pictures as well as text—a major advance in teleconferencing and electronic mail. This also lets an Apple "talk" to other kinds of computers, both large and small, at different locations, accessing the information that these store, and controlling special devices that are attached to them. In this way an Apple can be used to design pictures that are drawn and recorded in very high quality on special output devices that are shared by large numbers of users. This book is being produced by a novel method developed by the authors and their associates—transmitting from the Apple screen to an electronic typesetting machine to set text and graphics in a single pass. The potential for color as well as black-and-white printing and publishing, animation, and slide making, for business and education and entertainment, is vast.

All the programs in this book are written in an extended form of BASIC called Applesoft. An excellent description of this is *The Applesoft Tutorial* [Apple Product Number a2L0018 (030-0044-C)]. Refer to the Tutorial for backup if you need it. Chapter 8 provides a summary for readers skilled in other programming languages who want to make changes to the programs.

Chapter 2

Drawing some friends and foes

To get you started on Apple graphics, this chapter begins with the Applesoft instructions for drawing a simple face in the "immediate" mode. Making this wink and blink provides a first step in animation, and Cuthbert the Calculating Cur puts a little arithmetic in the picture. Sir Gruntgroan introduces the commands for high-resolution graphics and some simple sounds, and Cyrus the Sensitive Centipede illustrates another way to produce animation effects. The chapter ends with a simple program for constructing line drawings from the coordinates of their corners by point-to-point plotting.

2.1 DRAWING A FACE WITH LOW-RESOLUTION GRAPHICS

Pictures can be produced on the screen of an Apple by typing simple instructions on the keyboard. The "straight face" in Figure 2.1 illustrates these. The face is drawn on graph paper in Figure 2.2 to help explain the instructions.

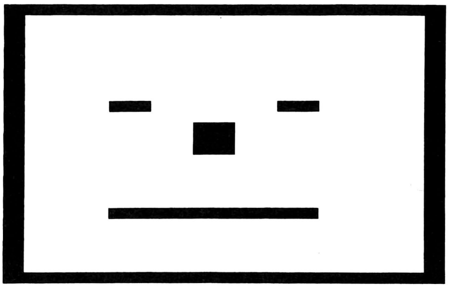


Figure 2.1

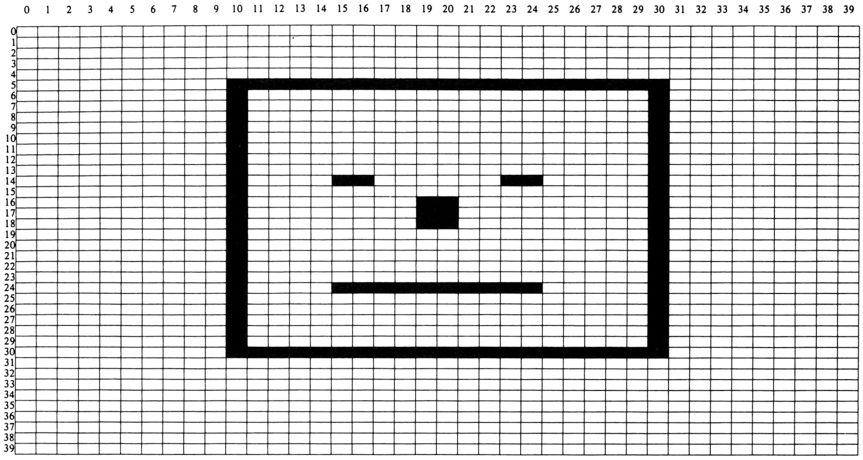


Figure 2.2

Instructions to draw the face are shown in the listing. The first instruction,

```
GR
```

puts the computer in the "low-resolution graphics" mode. This treats the screen rather like a sheet of graph paper with 40 boxes in each direction, numbered 0–39 as shown in Figure 2.2. The *y* coordinates are measured downward from the top of the screen. The second instruction,

```
COLOR=15
```

makes the pictures appear in white when using an ordinary monitor and also when the Apple is connected to a color display device. The third instruction,

```
HLIN 10,30 AT 5
```

draws the top line of the face. The letters HLIN stand for "horizontal line." These are followed by the *x* coordinates at the beginning and end of the line and, after the word AT, by its *y* coordinate. The next two instructions,

```
HLIN 15,16 AT 14
HLIN 23,24 AT 14
```

draw the left and right eyes. The nose is drawn by the instructions

```
VLIN 16,18 AT 19
VLIN 16,18 AT 20
```

The letters VLIN stand for "vertical line." These are followed by the *y* coordinates of the two ends of the line and, after the word AT, its x coordinate. The mouth and the sides and bottom of the face are drawn by the instructions

```
HLIN 15,24 AT 24
VLIN 5,30 AT 10
VLIN 5,30 AT 30
HLIN 10,30 AT 30
```

The complete set of instructions is

```
GR
COLOR=15
HLIN 10,30 AT 5
HLIN 15,16 AT 14
HLIN 23,24 AT 14
VLIN 16,18 AT 19
VLIN 16,18 AT 20
HLIN 15,24 AT 24
VLIN 5,30 AT 10
VLIN 5,30 AT 30
HLIN 10,30 AT 30
```

The face is made to smile in Figures 2.3 and 2.4 by adding a single box at each corner of the mouth with the instructions

```
PLOT 15,23
PLOT 24,23
```

The word PLOT is followed by the x and y coordinates of a single box.

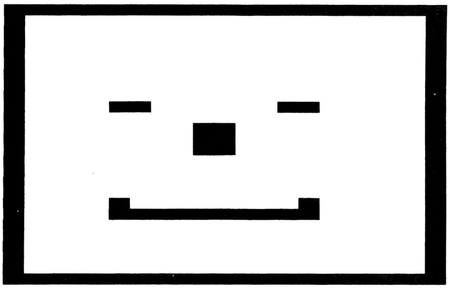


Figure 2.3

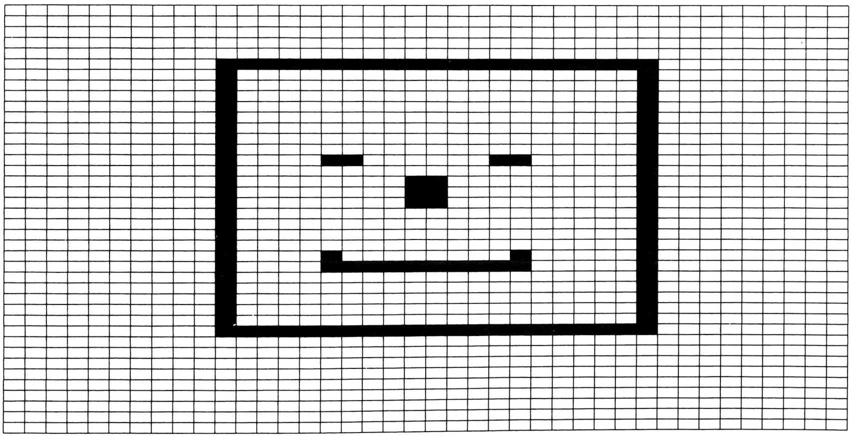


Figure 2.4

Suggested variations

1. Draw a series of vertical bars that are aligned at the top.
2. Draw a series of bars that are aligned at the bottom.
3. Draw some horizontal bars that are aligned (a) at the left, (b) at the right.
4. Block out some squares, triangles, lozenges, and other geometrical shapes, using vertical or horizontal lines.
5. Draw some other pictures on graph paper, and type the instructions to display these on the Apple's screen.
6. Get some patterns that have been drawn on squared paper—for example, for needlework or carpets, or designs that can be made with tiles, or bricks, or simple woodwork—and display these on the screen. Some readers may find it interesting to explore the variety of arts and crafts that use designs on squared paper.
7. See what happens when you use different numbers in the COLOR command, even if you only have a black-and-white screen.

2.2 A LITTLE WINKING AND BLINKING—ELEMENTARY ANIMATION

Now let's get some action. The eyes in the smiling face are enlarged in Figures 2.5. and 2.6. Each eye occupies 9 of the 40×40 rectangular boxes on the screen that can be lighted up in the low-resolution graphics mode. These are also called low-resolution *pixels* or *bricks*. Instructions to make the face appear on the screen, with its left eye alternately open and closed, constitute the BASIC program WINKFACE, shown in the listing at the end of the section.

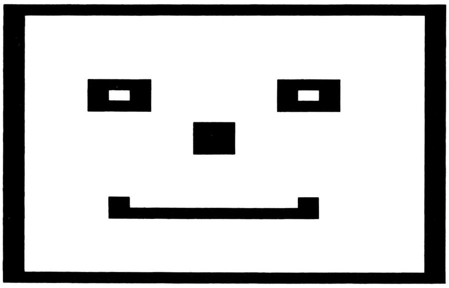


Figure 2.5

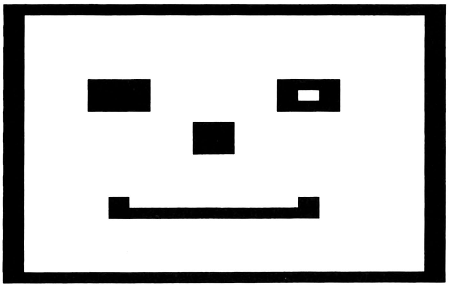


Figure 2.6

Lines 140–200 define the face with its eyes open, using some commands of low-resolution graphics already discussed. Line 220,

```
220 COLOR=15: PLOT 15,13
```

closes the left eye for a period determined by line 240:

```
240 FOR DE=1 TO 40: NEXT DE
```

The winking can be made slower or faster by increasing or decreasing the upper limit (40) in the FOR statement. Line 260

```
260 COLOR=0: PLOT 15,13
```

reopens the eye by changing the color of the pupil to the background color of the screen (that is, COLOR=0), for a time that is determined by the FOR loop in line 280:

```
280 FOR DE=1 TO 40: NEXT DE
```

Line 300,

```
300 GO TO 220
```

sends control back to the statement to reclose the eye.

Typing the word RUN makes the computer execute the program. This continues until CONTROL C is pressed (that is, until the CONTROL key and the C key are pressed simultaneously).

The faces in the preceding section can be produced by typing a series of instructions in the "immediate" mode because the pictures, once drawn, do not change. This book deals almost entirely with programs that produce pictures when they are executed. The reader should remember, however, that the immediate mode is very useful for sketching, and for adding to a picture that has been drawn by a program and frozen on the screen when a program ends.

Suggested variations

1. Make the right eye wink instead of the left.
2. Make both eyes wink in alternation.
3. Make both eyes blink together.
4. Make the mouth open and shut.
5. Make the eyes wink or blink 100 times. *Hint:* Use a FOR loop that begins

```
205 FOR I=1 TO 100
```

and change statement 300 correspondingly to

```
300 NEXT I
```

6. Create some animation effects based on the suggested variations in the preceding section.

The WINKFACE program

```
100  REM *** WINKFACE ***
110  REM *** USES LOW-RES GRAPHICS ***
120  GR : COLOR=15
130  REM *** DRAW FACE WITH OPEN EYES ***
140  HLIN 10,30 AT 5: HLIN 14,16 AT 14: HLIN 23,25 AT 14
150  VLIN 16,18 AT 19: VLIN 16,18 AT 20: HLIN 15,24 AT 24
160  VLIN 5,30 AT 10: VLIN 5,30 AT 30: HLIN 10,30 AT 30
170  PLOT 15,23: PLOT 24,23
180  HLIN 14,16 AT 12: HLIN 23,25 AT 12
190  PLOT 14,13: PLOT 16,13
200  PLOT 23,13: PLOT 25,13
210  REM *** CLOSE LEFT EYE ***
220  COLOR=15: PLOT 15,13
```

```
230  REM *** TIME DELAY WHILE IT IS CLOSED ***
240  FOR DE=1 TO 40: NEXT DE
250  REM *** OPEN LEFT EYE ***
260  COLOR=0: PLOT 15,13
270  REM *** TIME DELAY WHILE IT IS OPEN ***
280  FOR DE=1 TO 40: NEXT DE
290  REM *** REPEAT THE CYCLE ***
300  GOTO 220
```

2.3 GOING TO THE DOGS

The program DOGADD livens up some simple arithmetic. Cuthbert the Calculating Cur appears on the screen with a prompting message to the viewer that asks whether he should add, subtract, or multiply two numbers. After the viewer has replied, the program requests the numbers that are to be combined. When these have been provided, Cuthbert gives the answer by stamping his front paw the appropriate number of times. If the result is more than 100, he wags his tail for the hundreds part of the answer and stamps his foot for the rest. Figures 2.7A and 2.7B show Cuthbert with his foot down and raised, respectively.

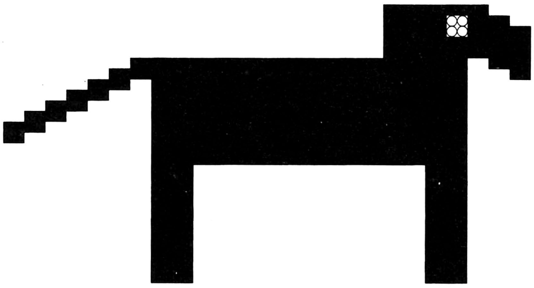


Figure 2.7A

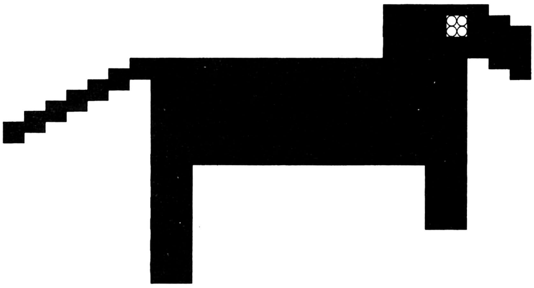


Figure 2.7B

The main variables in the program are

`X,Y`	The coordinates used to draw lines and points
`PL`	The index in the the cycles that use coordinates that are listed in the DATA statements
`IN$`	The word for the arithmetic operation requested by the user
`W$`	Its first letter
`A1,A2`	The numbers to be combined
`AN`	The answer
`H`	The number of hundreds in the answer (for example, if the answer is 245 then H is 2)
`AM`	The hundreds part of the answer (that is, 100×H)
`R`	The rest of the answer
`DE`	The index that controls a delay cycle
`LI`	The index that controls the foot-stamping cycle
`Z`	The index that controls the error message cycle

The program

1. Draws the dog in its rest position.
2. Prompts for the word that specifies the arithmetic operation, and stores this as the current value of the string variable IN$.
3. Uses the substring function

   ```
   LEFT$
   ```

   to test whether the first letter is A (for add), S (for subtract), or M (for multiply). An appropriate branch is taken, to request and store the numbers and to calculate the results.
4. Determines whether the result is more than 100; and calls the tail-wagging subroutine H times if it is. This erases and replots the tail in five successive positions going up and then going down by calling five further subroutines.
5. Determines whether the result is an exact multiple of 100; if it is not, the foot is stamped by erasing and redrawing the lower part the appropriate number of times.
6. Starts the process over again, from the beginning.

The instructions

`COLOR=12` and `COLOR=15`

are used before the body and the eye are drawn, to give different colors or, on a black-and-white screen, different shading patterns in the squares that are lighted up. For the full range of colors 0–15, these patterns can be seen easily by drawing a series of vertical bars, with the VLIN command.

The program also illustrates the use of the HOME, VTAB, HTAB, READ, and DATA statements.

Suggested variations

1. Make the dog stamp its rear paw for the tens part of the answer, and its forepaw for just the units.
2. Use different pictures to animate the results—for example, winking wombats, blinking baboons, ant-eating aardvarks with tongues that shoot out, open-mouthed oppossums, nodding newts, grinning griffons, laughing hyenas, and Cleo the camel that counts.
3. Devise ways for the picture to signal negative numbers, and use these.
4. Allow arithmetic expressions such as

   $1+2 \qquad 3-2 \qquad 2\times3$

   to be typed, with the "operator symbol," that is, + or − or ×, between the numbers. [*Hint:* (i) Store the entire arithmetic expression as a character string. (ii) Use a FOR loop and the MID$ function to test each character in the string, to find the operator symbol. (iii) Use the LEFT$ and RIGHT$ functions to separate the parts of the string that contain the numbers. (iv) Use the VAL function to set numerical variables to the values of these numbers.]
5. Give the animal a classical education and let the user type the numbers in Roman numerals.

6. Have the animal signal in Roman numerals as well (for example, Bilius Caesar's mare Nostra, which shuts its right and left eyes and its mouth for I, V, and X, respectively).
7. Bring it up to date with the new math, to use numbers that are typed in binary.
8. Make it signal the answers in binary (perhaps Cleo's cousin, Drusilla the dromedary, could use her two humps for this).
9. Put a hex on it, to accept hexadecimal input.
10. Widen the animal's horizon to deal with times, distances, areas, money, and other items that are expressed in mixed units.
11. A challenge to the mathematically minded—teach the animal to deal with fractions. (*Hint:* Find out about Euclid's method to compute greatest common divisors.)

The DOGADD program

```
100   REM *** DOGADD ***
110   REM *** USES LOW-RES GRAPHICS ***
120   HOME : GR : COLOR=12
130   REM *** DRAW THE DOG ***
140   FOR X=10 TO 11: VLIN 35,25 AT X: VLIN 35,25 AT X+13: NEXT X
150   FOR Y=24 TO 15 STEP -1: HLIN 10,24 AT Y: NEXT Y
160   FOR PL=1 TO 14: READ X,Y: PLOT X,Y: NEXT PL
170   DATA 9,15,9,16,8,16,8,17,7,17,7,18,6,18
180   DATA 6,19,5,19,5,20,4,20,4,21,3,21,3,22
190   FOR Y=10 TO 14: HLIN 21,25 AT Y: NEXT Y
200   VLIN 11,15 AT 26: VLIN 12,16 AT 27
210   COLOR=15: PLOT 24,12: PLOT 24,11
220   REM *** PROMPT FOR INPUT ***
230   INPUT "ADD, SUBTRACT OR MULTIPLY? ";IN$
240   INPUT "NUMBERS TO BE COMBINED: ";A1,A2
250   REM *** COMPUTE THE RESULT ***
260   W$=LEFT$(IN$,1)
270   IF W$="A" THEN AN=A1+A2: GOTO 310
280   IF W$="S" THEN AN=A1-A2: GOTO 310
290   IF W$="M" THEN AN=A1*A2: GOTO 310
300   GOSUB 800: GOTO 230: REM *** BAD INPUT ***
310   IF AN>0 THEN 360: REM *** BRANCH IF POSITIVE ***
320   IF AN<0 THEN  GOSUB 800: GOTO 230: REM *** BRANCH IF NEGATIVE ***
330   HOME : VTAB 21: HTAB 10: PRINT "0": REM *** RESULT IS 0 ***
340   FOR DE=1 TO 100: NEXT DE: GOTO 670
350   REM *** SPLIT THE RESULT INTO DIGITS ***
360   H=INT(AN/100):R=AN-H*100
370   IF H=0 THEN 580: REM *** NO HUNDREDS ***
380   REM *** WAG THE TAIL FOR HUNDREDS ***
390   FOR LO=1 TO H
400   COLOR=0: GOSUB 690: COLOR=12: GOSUB 710
410   FOR DE=1 TO 5: NEXT DE
420   COLOR=0: GOSUB 710: COLOR=12: GOSUB 730
```

```
430  FOR DE=1 TO 100: NEXT DE
440  COLOR=0: GOSUB 730: COLOR=12: GOSUB 750
450  FOR DE=1 TO 10: NEXT DE
460  COLOR=0: GOSUB 750: COLOR=12: GOSUB 770
470  FOR DE=1 TO 10: NEXT DE
480  COLOR=0: GOSUB 770: COLOR=12: GOSUB 750
490  FOR DE=1 TO 10: NEXT DE
500  COLOR=0: GOSUB 750: COLOR=12: GOSUB 730
510  FOR DE=1 TO 10: NEXT DE
520  COLOR=0: GOSUB 730: COLOR=12: GOSUB 710
530  FOR DE=1 TO 10: NEXT DE
540  COLOR=0: GOSUB 710: COLOR=12: GOSUB 690
550  HOME : VTAB 22: HTAB 10: PRINT LO*100
560  PRINT CHR$(7);: REM *** BELL CODE FOR BEEP ***
570  NEXT LO:AM=100*H
580  IF R=0 THEN 670: REM *** NO UNITS ***
590  REM *** FOOT STAMPS FOR UNITS ***
600  FOR LO=1 TO R: COLOR=0: FOR LI=35 TO 30 STEP -1
610  HLIN 23,24 AT LI: FOR DE=1 TO 5: NEXT DE,LI
620  PRINT : COLOR=12: FOR LI=30 TO 35: HLIN 23,24 AT LI
630  FOR DE=1 TO 5: NEXT DE,LI: HOME : VTAB 22
640  PRINT CHR$(7);: REM *** BELL CODE FOR BEEP ***
650  HTAB 10: PRINT LO+AM: NEXT LO
660  REM *** REPEAT CYCLE ***
670  GOTO 230
680  REM *** DRAW TAIL IN DIFFERENT POSITIONS ***
690  RESTORE : FOR PL=1 TO 14: READ X,Y: PLOT X,Y: NEXT PL
700  RETURN
710  HLIN 9,7 AT 15: HLIN 9,7 AT 16: HLIN 6,4 AT 17: HLIN 6,4 AT 18
720  RETURN
730  HLIN 9,4 AT 15: HLIN 9,4 AT 16
740  RETURN
750  HLIN 9,7 AT 16: HLIN 9,7 AT 15: HLIN 6,4 AT 14: HLIN 6,4 AT 13
760  RETURN
770  PLOT 9,15: PLOT 9,16: PLOT 8,15: PLOT 8,16: PLOT 7,15: PLOT 6,15
780  PLOT 7,14: PLOT 6,14: PLOT 5,14: PLOT 5,13: PLOT 4,13: PLOT 4,14
790  RETURN
800  REM *** USER GAVE INVALID INPUT ***
810  FOR Z=1 TO 3: FOR X=6 TO 0 STEP -1
820  COLOR=3: VLIN 39-X*3,40-X*3 AT 3+X
830  COLOR=0: VLIN 39-3*X,40-3*X AT 3+X
840  FOR DE=1 TO 4: NEXT DE
850  NEXT X,Z
860  RETURN
```

2.4 SIR GRUNTGROAN AND HIGH-RESOLUTION GRAPHICS

The program ROBOT displays a small robot (one of Sir Gruntgroan's robber band) in the firing line of a stream of arrows (Figure 2.8). When an arrow hits

its legs, the speaker in the Apple gives a short grunt and the legs vanish. When an arrow hits the top of the head, the robot is scalped. When an arrow hits the robot anywhere else, it disappears with a long groan. Another robot then appears, and the process continues until you press CONTROL C to stop. The vertical positions of the successive robots and the successive arrows are random.

Figure 2.8

The program uses the high-resolution graphics mode of the Apple. This treats the upper part of the screen as an array of square pixels that are much smaller than the pixels in low-resolution graphics. In the high-resolution mode there are 280 pixel positions across the screen (numbered 0–279) and 160 pixel positions going down from the top of the screen (numbered 0–159). This leaves room for four lines of text at the bottom of the screen. The graphics area can be extended downward to use this space too, giving 280×192 pixels, by a command that is described later.

The main variables in the program are

`XR,YR`	Coordinates of the top left corner of a robot's head
`X,Y`	Coordinates of the end of an arrow
`DE`	Index of a FOR loop that provides a delay between successive positions of the arrow
`H`	The address –16336 that is used in a PEEK command to make the speaker click

The program

1. Sets the coordinates of the robot, XR and YR, by using the random number function RND to compute a value between 0 and 1, and the INT function to extract the integer part of this random number times 60. The expression

```
INT(RND(1)*60)+20
```

thus provides a random integer between 20 and 80. The argument of the RND function must be a positive integer to produce different random numbers, but its value is otherwise immaterial.

2. Uses the commands

```
HGR: HCOLOR=3
```

to switch to the high-resolution graphics mode, and to display pictures in white on an ordinary or a color monitor.

3. Branches to a subroutine to draw the head and body of a robot, and another to draw its legs.
4. Uses the random number function RND and the integer function INT again, to form a random integer between 0 and 80, which makes the *y* coordinate

of the arrow shaft somewhere between 20 pixels above the top of the robot and 20 pixels below its feet:

```
Y=YR+INT(RND(1)*80)-20
```

5. Displays and erases the arrow in 8 successive positions across the screen, by means of a FOR loop. In each cycle, the instructions

   ```
   HCOLOR=3: GOSUB 550
   ```

   display the arrow in a new position, and the instructions

   ```
   X=X+10: HCOLOR=0: GOSUB 550
   ```

   erase it from its previous position.
6. Determines whether the arrow, which now has its tip in the same vertical line as the front of the robot, has hit the body or the scalp or the legs, and branches appropriately if it has.
7. Continues the flight of the arrow to the left of the screen if it missed, and repeats the arrow-firing process.
8. Produces the first part of a long groan by the instructions

   ```
   FOR LO=1 TO 30: X=PEEK(H): NEXT LO
   ```

   if there was a direct hit. These make the speaker click 30 times in rapid succession. Each click is produced by the PEEK function, when its argument is –16336. We shall have more to say about this function later. The same principle is then used to make the speaker repeat 3 rapid clicks 20 times, so producing a different tone. The final statement loops back to start the process again.
9. Erases the legs, if hit, by changing the color to the background and calling the subroutine that draws the legs. A short grunt is produced, which consists of 2 clicks, provided by references to the PEEK function, then a short delay caused by the null PRINT instruction, and then a single click, all repeated 10 times.

Suggested variations

1. Make the robot disappear in smaller pieces.
2. Start with a column of robots that are shot away one by one.
3. Have two columns of robots armed with crossbows, facing each other and taking turns at shooting.
4. Display a tower on one side of the screen and have a stream of projectiles shoot it away in pieces.
5. In the text area, display a running count of the number of arrows that have been fired at the current robot.
6. Also display the average number of arrows that it has taken to demolish successive robots in the current run.
7. Let the user vary the vertical range of the arrows, and see how this relates to the average number of shots needed to eliminate a robot.
8. Let the user vary the size of the robot too, and see how this affects the average.

The ROBOT program

```
100  REM *** ROBOT ***
110  HOME : REM *** CLEARS THE SCREEN ***
120  H=-16336: REM *** PEEKVALUE FOR CLICK ***
130  REM *** USES HI-RES GRAPHICS ***
140  HGR : HCOLOR=3
150  REM *** DRAW A ROBOT ***
160  XR=20:YR=INT(RND(1)*60)+20
170  GOSUB 410: GOSUB 510
180  REM *** FIRE AN ARROW ***
190  Y=YR+INT(RND(1)*80)-20
200  FOR X=200 TO 40 STEP -20
210  HCOLOR=3: GOSUB 550
220  HCOLOR=0:X=X+10: GOSUB 550
230  FOR DE=1 TO 7: NEXT DE
240  NEXT X
250  IF Y>YR AND Y<YR+30 THEN 340: REM DIRECT HIT
260  IF Y>YR AND Y<YR+40 THEN 370: REM HIT ON LEGS
270  FOR X=30 TO 10 STEP -20: REM *** MISSED ***
280  HCOLOR=3: GOSUB 550
290  X=X+10: HCOLOR=0: GOSUB 550
300  FOR DE=1 TO 7: NEXT DE
310  NEXT X
320  X=10: GOSUB 550: GOTO 190
330  REM *** GROAN ***
340  FOR LO=1 TO 30:X=PEEK(H): NEXT LO
350  FOR LO=1 TO 20:X=PEEK(H)+PEEK(H)+PEEK(H): NEXT
360  GOTO 140
370  HCOLOR=0: GOSUB 510: FOR LO=1 TO 10
380  X=PEEK(H)+PEEK(H): PRINT :X=PEEK(H): NEXT LO
390  GOTO 270
400  END
410  REM *** DRAW HEAD AND BODY ***
420  HPLOT XR,YR TO XR+10,YR TO XR+10,YR+8 TO XR,YR+8 TO XR,YR
430  HPLOT XR+3,YR+2 TO XR+3,YR+3
440  HPLOT XR+5,YR+5 TO XR+10,YR+5
450  HPLOT XR+3,YR+8 TO XR+3,YR+11
460  HPLOT XR-2,YR+11 TO XR+12,YR+11 TO XR+12,YR+30
470  HPLOT XR+12,YR+30 TO XR-2,YR+30 TO XR-2,YR+11
480  HPLOT XR+4,YR+16 TO XR+15,YR+24
490  HPLOT XR+12,YR+13 TO XR+16,YR+10
500  RETURN
510  REM *** DRAW LEGS ***
520  HPLOT XR+4,YR+30 TO XR-2,YR+40
530  HPLOT XR+4,YR+30 TO XR+10,YR+40
540  RETURN
550  REM *** DRAW ARROW ***
560  HPLOT X,Y TO X-10,Y: HPLOT X-6,Y-3 TO X-10,Y TO X-6,Y+3
570  RETURN
```

2.5 CENTIPEDES AND CHORUS LINES

The CENTIPEDE program draws little centipedes like Cyrus and his cousins (Figure 2.9), and makes them run across the screen.

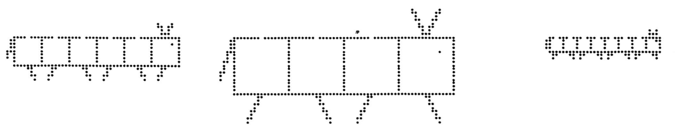


Figure 2.9

The first centipede has 6 segments, with a tail that closes the rear segment, and an eye and antennas to complete the one at the front (Figure 2.10). The second and third centipedes have 4 and 8 segments, respectively.

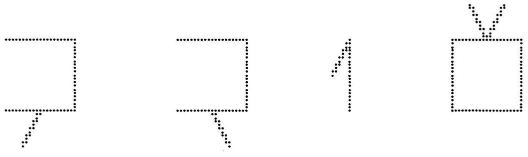


Figure 2.10

The program

1. Prompts the user for the details of a centipede and its path
2. Draws the centipede at the left of the screen
3. Repeatedly erases the tail, rear segment, eye, and antennas, and then redraws the tail and another segment at the front and puts an eye and antennas on this
4. Complains if the centipede bangs its head

The prompting message requests

1. The size of the segments, specified by the number of pixels in each side divided by 4 (this must be a whole number)
2. The number of segments in the centipede
3. The run length, that is, how many steps it will take across the screen
4. Its vertical position on the screen

A typical set of values is 2, 5, 25, 60.

The geometry is very simple (Figure 2.11). The main variables in the program are as follows:

`A`	The size of the segments (that is, 1/4 of the length of a side)
`SM`	The number of segments
`PM`	The run length
`Y0`	The *y* value of the back of the centipede (that is, the top of the segments)
`X0`	The *x* value of the top left corner of the segment that is being drawn or erased

XT	The *x* value of the top left corner of the rear segment
FT	The horizontal distance of the foot from the left edge (A and 3A alternately in successive segments)

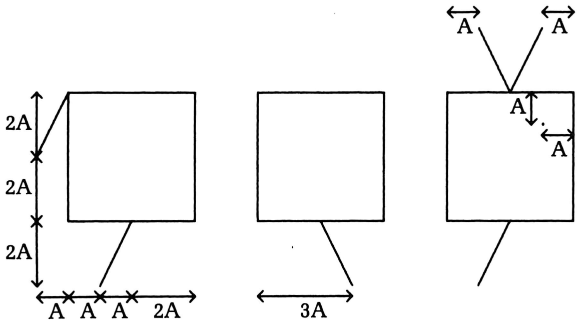


Figure 2.11

The program calls one subroutine to draw (or erase) a segment, a second subroutine to draw or erase the tail, and a third subroutine to draw or erase the eye and antennas.

By using these three building blocks, relatively little is erased and redrawn in each step. This principle of erasing from one end and redrawing at the opposite end can be used in a variety of animations.

Suggested variations

1. The centipede starts and ends "offstage."
2. Its speed can be varied by the user.
3. Its speed changes as it runs.
4. It runs from right to left.
5. A vertical centipede runs up and down the screen.
6. When one centipede runs off, another runs on, with random variations of size, speed, and direction.
7. Several centipedes are onstage together, moving in unison or having a race.
8. Keep count of the number of times the centipedes bang their heads, and vary the message or other action accordingly.
9. After the user types the data already described, the program asks for (a) the length of the centipede in screen units or (b) the distance it travels, and tells the user whether the response is correct (to reinforce elementary multiplication).
10. Use the same principle to make a chorus line of stick figures dance across the screen, with the leader wearing a hat.
11. Make a procession of robots march with the leader carrying a flag.
12. Design other processions and similar animation effects.

The CENTIPEDE program

```
100  REM *** CENTIPEDE ***
110  REM *** PROMPT FOR INPUT ***
```

```
 120  TEXT : HOME : VTAB 21
 130  PRINT "SIZE (CAN BE 1-6, NOW IS ";A;"): ";: INPUT "";A
 140  PRINT "LENGTH (CAN BE 1-10, NOW IS ";SB;"): ";: INPUT "";SB:SM=SB*2
 150  PRINT "RUN (CAN BE 1-30, IS NOW ";PM;"): ";: INPUT "";PM
 160  PRINT "POSITION (CAN BE 10-150, IS NOW ";Y0;"): ";: INPUT "";Y0
 170  A2=A+A:A4=A2+A2:A3=A2+A
 180  XT=A4:X0=A4:X4=X0+A4
 190  Y1=Y0+A:Y2=Y0+A2:Y3=Y2+A
 200  Y4=Y0+A4:Y6=Y4+A2:YA=Y0-A2:FT=A
 210  REM *** DRAW THE CENTIPEDE - START WITH TAIL ***
 220  HGR : HCOLOR=3: GOSUB 490
 230  REM : DRAW BLOCK 2 FOR ALL THE SEGMENTS
 240  FOR SN=1 TO SM
 250  GOSUB 520
 260  REM : MOVE THE CORNERS AND THE FOOT
 270  X0=X4:X4=X0+A4:FT=A4-FT
 280  NEXT SN
 290  REM *** DRAW THE EYE ***
 300  GOSUB 560
 310  REM *** MAKE IT RUN ***
 320  FOR P=1 TO PM
 330  REM *** BRANCH IF ERROR WILL OCCUR ***
 340  IF X4>279 THEN 1000
 350  REM *** ERASE THE TAIL ***
 360  HCOLOR=0: GOSUB 490
 370  REM : ERASE THE END SEGMENT
 380  XP=X0:X0=XT:X4=X0+A4: GOSUB 520:X0=XP:X4=X0+A4
 390  REM *** ERASE THE EYE ***
 400  GOSUB 560
 410  REM *** DRAW THE NEW TAIL ***
 420  HCOLOR=3:XT=XT+A4: GOSUB 490
 430  REM *** DRAW THE NEW FRONT SEGMENT AND EYE ***
 440  GOSUB 520:X0=X4:X4=X0+A4: GOSUB 560
 450  REM *** REVERSE THE DIRECTION OF THE LEG ***
 460  FT=A4-FT
 470  NEXT P
 480  GOTO 120
 490  REM *** DRAW BLOCK 1 *** (TAIL & SIDE IT IS ON) ***
 500  HPLOT XT-A,Y3 TO XT,Y0 TO XT,Y4
 510  RETURN
 520  REM *** DRAW BLOCK 2 *** (3 SIDES OF SEGMENT AND LEG)
 530  HPLOT X0,Y0 TO X4,Y0 TO X4,Y4 TO X0,Y4
 540  HPLOT X0+A2,Y4 TO X0+FT,Y6
 550  RETURN
 560  REM *** DRAW BLOCK 3 *** (THE EYE AND ANTENNAE) ***
 570  HPLOT X0-A,Y1
 580  HPLOT X0-A3,YA TO X0-A2,Y0 TO X0-A,YA
 590  RETURN
1000  REM *** ERROR ***
1010  TEXT : HOME : FLASH : VTAB 12: HTAB 10
```

```
1020  PRINT "DON'T BANG MY HEAD!": NORMAL
1030  FOR DE=1 TO 1000: NEXT DE: GOTO 120
```

2.6 PACING THINGS OUT

The program DRAW1 lets you make a point-to-point drawing on the screen. It displays axes along the edges of the screen in the graphics mode, with tick marks at 10-pixel intervals, and the main menu in the text mode:

```
  1 MOVE ABSOLUTE
  2 MOVE RELATIVE
  3 DRAW ABSOLUTE
  4 DRAW RELATIVE
  5 ERASE ABSOLUTE
  6 ERASE RELATIVE
  7 CLEAR SCREEN
  8 QUIT
WHICH OPTION?
```

The program acts as if you could

1. Move a pen to wherever you want on the screen, without making a mark.
2. Draw a straight line with the pen, from wherever it is to another position. This serves as its starting point for the next command.
3. Turn the pen into an eraser, and erase in a straight line from wherever it is to another position on the screen.

You can specify the end position of a MOVE, DRAW, or ERASE command in coordinates that are measured either

1. From the upper left corner of the screen or
2. From the position of the pen when the command is given

These are called absolute and relative coordinates, respectively.

The program clears the screen, plots axes with tick marks, switches to the text mode and displays the main menu. You select a command by typing its option number (for example, 3 for DRAW ABSOLUTE). The program then switches back to the graphics mode and prompts, at the bottom of the screen, for the absolute or relative coordinates of the next end point. After you respond, the program performs the action you requested, switches back to the text mode and displays the main menu again. You continue in this way until the picture is drawn, with the screen displaying the menu and the growing picture in alternation. Here are two simple examples of pictures that were produced in this way. Figure 2.12, a cube, was produced by the commands

1, 140, 80;   4, 40, 0;   4, 0, 40;   4, −40, 0;   4, 0, −40;
4, 20, −20;   4, 40, 0;   4, 0, 40;   4, −40, 0;   4, 0, −40;
2, 40, 0;   4, −20, 20;   2, −40, 40;   4, −20, −20;   2, 40, 0;
4, −20, 20.

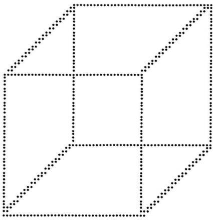


Figure 2.12

Figure 2.13, a turkey, was produced by the following commands:

```
1,   6,  42;   3,  32,  20;   3,  19,  88;   3,  17,  40;   1,  32,  20;
3,  42,  59;   1,   6,  42;   3,  36,  35;   1,  24,  62;   3, 136,  46;
3, 158,  60;   3, 101,  78;   3, 136,  46;   3,  96,  21;   3,  74,  55;
1, 101,  78;   3,  96,  83;   3, 111,  87;   3,  38,  92;   3,  24,  62;
3,  96,  83;   1,  51, 103;   3,  63,  90;   3,  55, 107;   3,  51, 103;
3,  40, 112;   3,  55, 107;   1,  64, 103;   3,  73,  90;   3,  71, 107;
3,  64, 103;   3,  52, 112;   3,  71, 107;   1,  26,  30;   3,  26,  30.
```

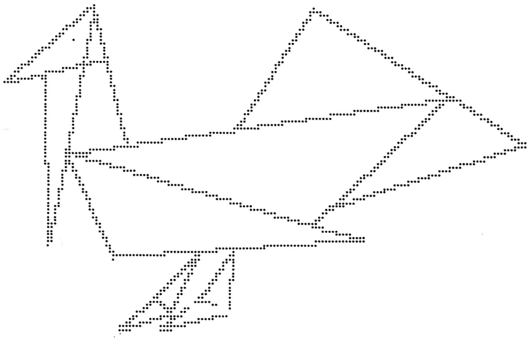


Figure 2.13

The main variables in the program are as follows:

`XT,YT`	The x and y values of the tick marks on the horizontal and vertical axes
`XO,YO`	The coordinates of the previous point that was plotted
`XN,YN`	The coordinates of the next point to be plotted
`XC,YC`	The relative coordinates of the next point
`OP`	The option number that has been specified
`TI$(8)`	The character strings that specify the options

In the Apple, different areas of memory store what is typed in the text mode and what is plotted in the high-resolution graphics mode. The command HGR erases the high-resolution graphics area, and switches the screen to it. The command TEXT erases the text area, and switches the screen back to this area.

The program follows the general course of action that has been described. It reads the descriptions of the options from a DATA statement into the array TI$ —this is shorter than using eight assignment statements. When the Apple is in the text mode, the statements

```
POKE -16297,0: POKE -16304,0
```

switch back to high-resolution graphics, without erasing what was on the screen prior to the previous switch to the text mode.

Suggested variations

1. Let the user type the length of a line to be drawn or erased, and its angle from the horizontal, as another option.
2. Allow the user to erase the axes as another option.
3. Let the user save the data that defines a set of drawings in a text file that can be read to redisplay the drawings subsequently (see Section 3.2).
4. Display a blinking dot or a small shape as a cursor to show the current pen position.
5. Use a cursor and different keys (a) to move it up, down, left, and right, (b) to make its current position the start of a line to be drawn or erased, (c) to make its current position the end of a line to be drawn, (d) to make its current position the end of a line to be erased, and (e) to cancel the effect of (b).
6. Extend this tactic to provide "rubber banding." In this process the computer draws a straight line from a fixed point that was set by the cursor to wherever the cursor is moved next. When the cursor is moved again, the beginning of the line remains fixed, but the other end moves with the cursor, and the line is redrawn accordingly.
7. Let the user save the data in an array, and then draw the same outline several times at positions for which the program prompts.
8. Let the user scale the pictures, after the data has been stored in an array, by different amounts in the *x* and *y* directions. This would produce turkeys like the following (Figure 2.14) from the data used for Figure 2.13.

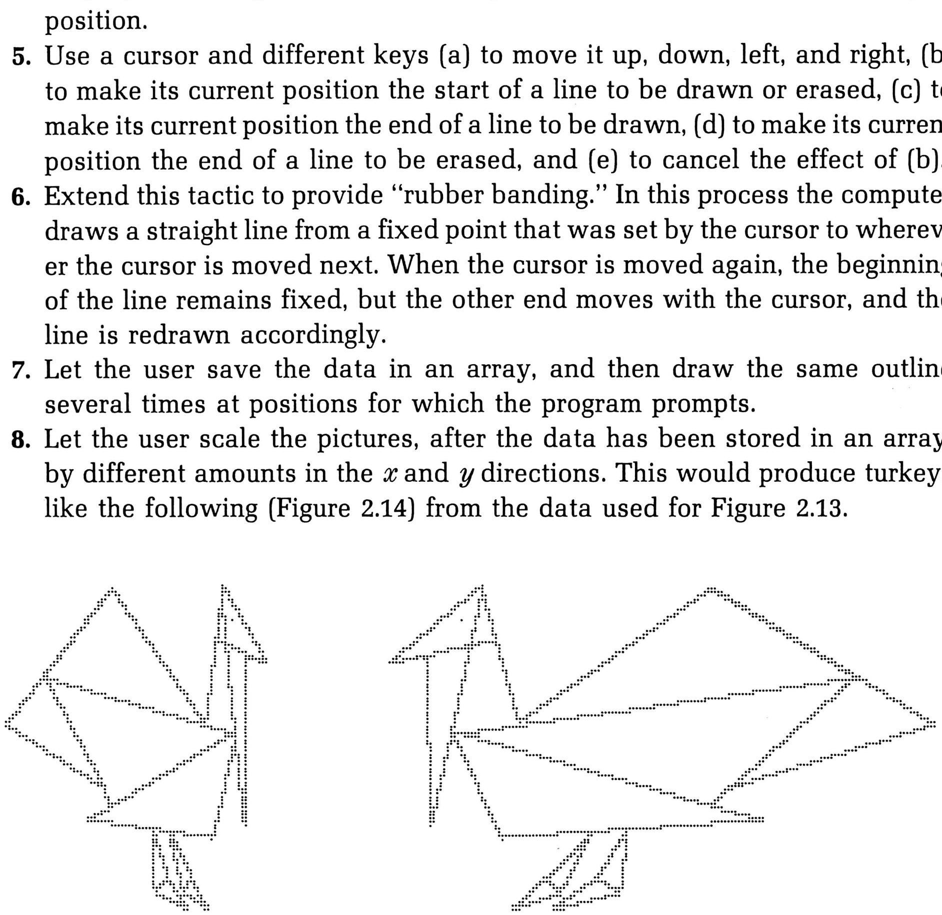


Figure 2.14

The DRAW1 program

```
100   REM *** DRAW1-USES ENDPOINT COORDINATES ***
110   LOMEM: 16384: REM *** AVOID MEMORY CONFLICT ***
120   ONERR  GOTO 650: REM *** BRANCH ON ERROR CONDITION ***
130   HOME : HGR : HCOLOR=3: REM *** HI-RES GRAPHICS ***
140   REM *** DRAW AXES AND TICK MARKS ***
150   HPLOT 0,0 TO 279,0 TO 279,159 TO 0,159 TO 0,0
```

```
160  FOR XR=0 TO 279 STEP 10
170  HPLOT XR,0 TO XR,3: HPLOT XR,156 TO XR,159
180  NEXT XR
190  FOR XR=0 TO 279 STEP 50
200  HPLOT XR,0 TO XR,5: HPLOT XR,154 TO XR,159
210  NEXT XR
220  FOR YR=0 TO 159 STEP 10
230  HPLOT 0,YR TO 3,YR: HPLOT 276,YR TO 279,YR
240  NEXT YR
250  FOR YR=0 TO 159 STEP 50
260  HPLOT 0,YR TO 5,YR: HPLOT 274,YR TO 279,YR
270  NEXT YR
280  REM *** INITIALIZE ALL VARIABLES ***
290  X0=0:Y0=0
300  FOR L=1 TO 8: READ TI$(L): NEXT L
310  DATA MOVE ABSOLUTE,MOVE RELATIVE,DRAW ABSOLUTE,DRAW RELATIVE
320  DATA ERASE ABSOLUTE,ERASE RELATIVE,CLEAR SCREEN,QUIT
330  REM *** DISPLAY MAIN MENU ***
340  TEXT : HOME : HTAB 16: PRINT "OPTIONS"
350  VTAB 4: FOR L=1 TO 8: HTAB 6: PRINT L;".  ";TI$(L): NEXT L
360  PRINT : HTAB 11: INPUT "WHICH OPTION? ";OP
380  IF OP<1 OR OP>8 THEN 340
390  IF OP>6 THEN 620
400  REM *** SWITCH TO HI-RES GRAPHICS DISPLAY ***
410  POKE -16297,0: POKE -16304,0
420  REM *** PROMPT FOR DATA ***
430  VTAB 21: PRINT TI$(OP)
440  PRINT "OLD COORDINATES ARE ";X0;",";Y0
450  IF OP=2 OR OP=4 OR OP=6 THEN 510
460  REM *** ABSOLUTE OPERATION ***
470  INPUT "CHANGE COORDINATES TO ";XN,YN
480  IF XN>=0 AND XN<=279 AND YN>=0 AND YN<=159 THEN 550
490  PRINT "ILLEGAL VALUE(S)": FOR LO=1 TO 1000: NEXT LO
500  VTAB 23: CALL -958: GOTO 450
510  REM *** RELATIVE OPERATION ***
520  INPUT "CHANGE COORDINATES BY ";XC,YC
530  XN=X0+XC:YN=Y0+YC
540  GOTO 480
550  REM *** DRAW OR ERASE A LINE ***
560  IF OP<3 THEN 590
570  HCOLOR=3*(OP<5)
580  HPLOT X0,Y0 TO XN,YN TO X0,Y0
590  X0=XN:Y0=YN
600  FOR DE=1 TO 1000: NEXT DE
610  GOTO 340
620  REM *** CLEAR SCREEN OR QUIT ***
630  IF OP=7 THEN  CLEAR : RESTORE : GOTO 130
640  IF OP=8 THEN  END : REM *** TERMINATES PROGRAM OPERATION ***
650  RESUME
```

Chapter 3

Building blocks and shape tables

Pictures and diagrams contain portions that are repeated—within the same illustration, or in a series of illustrations, or in successive frames of an animation—in countless situations. This chapter describes some simple ways to define, display, and store the data for portions of pictures, and for the grouping of these portions together in the complete pictures. The first two sections use some programming tactics that apply to many other problems of hierarchical structuring as well. The further sections use the special "shape table" provisions of the Apple.

3.1 A BACKDROP FOR *SOUTH PACIFIC*

The program ALOHA1 draws scenes of desert islands, with or without palm trees and birds flying overhead (Figures 3.1 and 3.2). Many other panoramas can be drawn easily, using the tactics of this program.

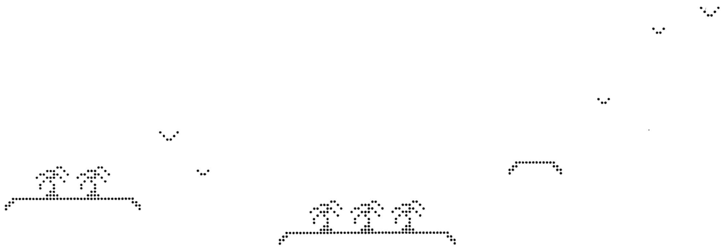


Figure 3.1

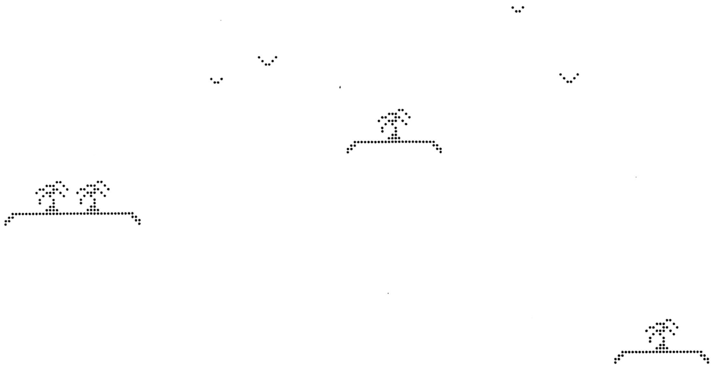


Figure 3.2

Each ALOHA1 panorama consists of five building blocks that you can repeat and arrange. These are in Figure 3.3 a small bird (block 1), a large bird (block 2), the left and right sides of an island (blocks 3 and 4, respectively), and a middle part with a palm tree (block 5).

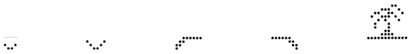


Figure 3.3

The program clears the graphics screen and prompts the user by displaying the message

```
OPTIONS:  1  DISPLAY A BUILDING BLOCK
          2  ERASE SCREEN
          3  QUIT
WHICH OPTION?
```

When you type 1, the program responds by prompting for

1. The number that identifies the building block (that is, 1 for the small bird, 2 for the large bird, and so forth)
2. The horizontal space between the right edge of the last building block that was displayed (or the left edge of the screen if it was just cleared) and the left edge of the new building block
3. Its vertical position on the screen

Figure 3.1 was produced by typing 1, in alternation with each of the following sets of three numbers:

3, 0, 70; 5, 0, 70; 5, 0, 70; 4, 0, 70; 2, 5, 50; 1, 5, 60;
3, 20, 80; 5, 0, 80; 5, 0, 80; 5, 0, 80; 4, 0, 80; 3, 15, 60;
4, 0, 60; 1, 10, 40; 1, 12, 20; 2, 10, 15.

The first four sets of three numbers display the first island. The next two sets display the first two birds. The next five sets display the second island. The next two sets display the treeless island, and the last three sets display the last three birds.

The horizontal space before a building block can be negative. One block thus can be placed above another, as in Figure 3.4, which also contains hills and valleys.

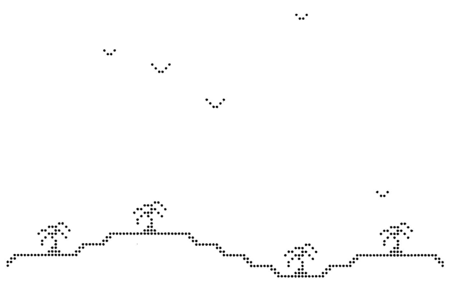


Figure 3.4

The program contains separate subroutines to draw each building block. The main variables are

`ID`	The number that identifies a building block
`XR`	The *x* coordinate of the left edge of the building block
`YR`	The *y* coordinate of its upper edge
`HS`	The horizontal space between a building block and its predecessor (or the left edge of the screen)

When you have typed the option to display a building block, the values of ID, HS, and YR are requested and stored, and the program branches on ID to the appropriate subroutine. Each of these consists of a succession of HPLOT statements, which draw a succession of horizontal lines and individual pixels that form the building block at the appropriate position. XR is increased by the width of the building block.

To write a program like this, the building blocks can be drawn on graph paper first, and the HPLOT commands designed accordingly (Figures 3.5A–3.5E).

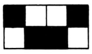


Figure 3.5A

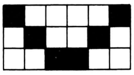


Figure 3.5B

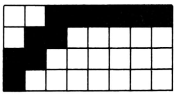


Figure 3.5C

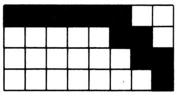


Figure 3.5D

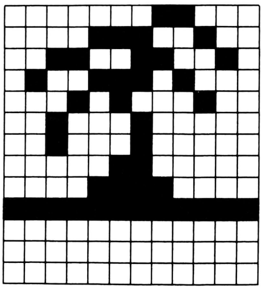


Figure 3.5E

The building blocks can be designed much more rapidly on the screen itself, using the program DESIGN1 that is described two sections hence. You can change the building blocks in a program like ALOHA1 by altering the subroutines. The DESIGN2 program in Section 3.4 lets you change the building blocks without altering any instructions, but it is somewhat longer. The special shape table feature of the Apple software provides yet another way to handle this kind of situation, with considerable benefits in certain circumstances. This feature is presented in sections 3.5–3.8. It is useful, all the same, to see examples of the different approaches to the problem.

Suggested variations

1. Expand ALOHA1 to include treeless and sloping sections of an island, and also boats and huts.
2. Design the building blocks for some other sets of panoramas, processions, armadas, and so forth, and write the corresponding programs.
3. Allow the user to erase a building block that is on the screen.
4. Allow the user, when the horizontal space and vertical positions are constant, to type these once, followed by the number of building blocks to be displayed and their identifying numbers.
5. Display successive building blocks downward from the top of the screen by giving a horizontal position and vertical space instead of a vertical position and horizontal space for each.

The ALOHA1 program

```
100  REM *** ALOHA1-DRAWS SOUTH SEAS PANORAMA ***
110  REM *** USES HI-RES GRAPHICS ***
200  HGR : HCOLOR=3
210  HOME : VTAB 21
220  PRINT "1 DISPLAY A BUILDING BLOCK": PRINT "2 ERASE SCREEN"
230  PRINT "3 QUIT"
240  HTAB 6: INPUT "WHICH OPTION? ";IZ
250  ON IZ GOTO 300,200,4000
260  GOTO 210
300  REM *** DISPLAY A BUILDING BLOCK"
310  HOME : VTAB 21
320  INPUT "ID#, HORIZONTAL SPACE, AND VERTICAL POSITION: ";ID,HS,YR
330  XR=XR+HS
340  ON ID GOSUB 400,500,600,700,800
350  GOTO 210
400  REM *** DISPLAY A SMALL BIRD ***
410  HPLOT XR,YR-1: HPLOT XR+1,YR TO XR+2,YR: HPLOT XR+3,YR-1
420  XR=XR+4: RETURN
500  REM *** DISPLAY A LARGE BIRD ***
510  HPLOT XR,YR-2: HPLOT XR+1,YR-1: HPLOT XR+5,YR-2
520  HPLOT XR+2,YR TO XR+3,YR: HPLOT XR+4,YR-1
530  XR=XR+6: RETURN
600  REM *** DISPLAY THE LEFT SIDE OF AN ISLAND ***
610  HPLOT XR,YR TO XR+2,YR-3 TO XR+7,YR-3
```

```
 620  XR=XR+8: RETURN
 700  REM *** DISPLAY THE RIGHT SIDE OF AN ISLAND ***
 710  HPLOT XR,YR-3 TO XR+5,YR-3 TO XR+7,YR
 720  XR=XR+8: RETURN
 800  REM *** DISPLAY A PALM TREE ***
 805  HPLOT XR,YR-3 TO XR+11,YR-3
 810  HPLOT XR+4,YR-4 TO XR+7,YR-4
 815  HPLOT XR+5,YR-5 TO XR+6,YR-5
 820  HPLOT XR+6,YR-6: HPLOT XR+6,YR-7
 825  HPLOT XR+2,YR-6: HPLOT XR+2,YR-7
 830  HPLOT XR+3,YR-8: HPLOT XR+5,YR-8
 835  HPLOT XR+9,YR-8: HPLOT XR+1,YR-9
 840  HPLOT XR+4,YR-9 TO XR+6,YR-9: HPLOT XR+8,YR-9
 845  HPLOT XR+2,YR-10 TO XR+3,YR-10
 850  HPLOT XR+6,YR-10 TO XR+7,YR-10
 855  HPLOT XR+10,YR-10: HPLOT XR+4,YR-11 TO XR+6,YR-11
 860  HPLOT XR+9,YR-11: HPLOT XR+7,YR-12 TO XR+8,YR-12
 870  XR=XR+12: RETURN
 880  HPLOT XR+4,YR-9 TO XR+6,YR-9: HPLOT XR+8,YR-9
 890  HPLOT XR+2,YR-10 TO XR+3,YR-10
4000  END
```

3.2 PANORAMAS FOR POSTERITY

Suppose you want to construct some panoramas by typing the data you would use for ALOHA1 and then have the computer redisplay these scenes without having to type the data all over again, after you used it for some other work. ALOHA2 lets you save the data as a "text file" on disk in one run. You can make the computer read and use this file in later runs whenever you want.

Saving information in text files on disk is enormously important, for example, when dealing with

1. Business data that you update, total, and refer to periodically
2. Data that express some designs that you developed interactively
3. Output of one program that you want to use as input to another
4. Lengthy lists of constants that several programs need to use

The last part of ALOHA2 is a prototype that you can use for these purposes.

The program begins by prompting:

```
1. CREATE A PANORAMA
2. SAVE A PANORAMA
3. READ A PANORAMA
4. QUIT
WHICH OPTION?
```

Make a start by using it in the same way as ALOHA1, to create a panorama. Type 1, and you get the secondary menu:

```
1. DISPLAY BUILDING BLOCK
2. ERASE SCREEN
3. RETURN TO MAIN MENU
```

Carry on as before. Each time you type 1, the program prompts

```
ID #, HORIZONTAL SPACE, AND VERTICAL POSITION:
```

After you have typed these, a building block appears on the screen. You can start the panorama afresh, by typing 2. When you want to save a panorama, type 3 to return to the main menu. Then type 2, and the program prompts

```
NAME OF FILE TO BE SAVED:
```

You can

1. Start a file with a name that is not currently in use, for any of the files already on the disk. Or
2. Start a file with the name of a file that already exists—thereby superseding it (irretrievably). Or
3. Append data to an existing file.

Type the file name that you want to use. The program prompts

```
TYPE B TO BEGIN OR A TO APPEND:
```

Type B or A for the alternative that you want. The data for the current panorama is saved, and the program repeats the main prompt, for you to continue as you wish.

When a panorama file has been saved on disk, the program automatically displays the panoramas if you type 3 in response to the main prompt. This triggers

```
NAME OF FILE TO BE READ:
```

You then type the appropriate name. The program pauses when each panorama is on the screen, then erases it and displays the next.

The main variables in ALOHA2 are

OP	Option chosen from main menu
IZ	Option chosen from secondary menu
ID	Identification number of a building block
HS	Horizontal space before building block
XR	x position of building block
YR	y position of building block
ID(100)	Array to store the ID values of the successive building blocks in a panorama
HS(100)	Array to store the HS values of the successive building blocks in a panorama
YR(100)	Array to store the YR values of the successive building blocks in a panorama
IB	Index of these arrays—takes value 1, 2, ... for successive building blocks in a panorama, when it is being created; IB also is the number of building blocks in a panorama when it is complete
LO	Index of these arrays in the cycle that writes the data that define a panorama into a text file—takes values 1–IB
RE	Index of these arrays in the cycle that reads the data that defines a panorama from a text file

`TI$`	Name of the text file to be written or read
`DE`	Index of a FOR loop to provide a delay when a panorama has been read
`D$`	The CONTROL D code—this has the ASCII value 4

The variables IZ, ID, HS, XR and YR have the same role as in ALOHA1. The program begins by displaying the main menu. It uses a READ statement in a FOR loop to get the words that describe the successive options from a DATA statement. This is a useful tactic. It avoids long sequences of PRINT statements, and lets you change the wording simply by changing the DATA statement. After a few more statements that branch on the main menu option, the whole of ALOHA1 from statement 200 is incorporated in the new program with just the change of the GO TO address in statement 250, to repeat the main cycle, and two small insertions:

```
205 XR=0: IB=0
```

to reinitialize XR and IB at the start of a new panorama, and

```
325 IB=IB+1: ID(IB)=ID: HS(IB)=HS: YR(IB)=YR
```

This stores the values that have been typed of ID, HS, and YR in the arrays ID(100), HS(100), and YR(100), for later use when saving the panorama. Note that IB is reset to 0 by statement 205 whenever a panorama is begun.

The statements to write and read text files come at the end of the program. These follow the rules in the DOS manual in a direct way (see Section 8.1).

Thus in an Applesoft program to write a text file, the commands

```
OPEN file-name
WRITE file-name
```

are sent to the system, to open the file for writing. This is done by the Applesoft statements:

```
D$=CHR$(4)
PRINT D$;"OPEN ";TI$
PRINT D$;"WRITE ";TI$
```

The CONTROL D code provided by D$ makes the system interpret the rest of the PRINT statement as a character string that spells out a system command. After the file has been opened for writing, successive items are written into it by ordinary PRINT statements. At the end, the file is closed by sending the command

```
CLOSE file-name
```

to the system by another PRINT statement that begins with a CONTROL D code. A file is opened for reading by the commands

```
OPEN file-name
READ file-name
```

These are sent to the system by PRINT statements that begin with a CONTROL D code. The successive items are read by ordinary INPUT statements. At the end the file is closed, in just the same way as when writing. To append data to a file, the commands

```
APPEND file-name
WRITE file-name
```

are sent to the system, using the same tactic.

The data that describe a panorama consist simply of IB, the number of building blocks that it contains, and the ID, HS, and YR values of these. Statement 2160 writes these to disk. Correspondingly, when the file is read, lines 3060–3100 input IB for the first panorama, and then input and use the appropriate number of ID, HS, and YR values from the file. These are used to display building blocks just as if you were typing them. The program then provides a short delay, and repeats the process for each of the panoramas in the text file that is being read.

The ONERR statement in line 3030 makes the program bypass faulty input, and line 3520 closes the file to avoid damaging it in error situations.

Suggested variations

1. Combine these principles to use text files with the variations suggested for ALOHA1.
2. Allow the user to display the *n*th panorama in a text file. Make the program (a) prompt for the value of *n* and (b) display the appropriate panorama without the previous panoramas in the file appearing on the screen first.

The ALOHA2 program

```
100  REM *** ALOHA2-SAVES AND RETRIEVES PANORAMAS ***
110  DIM ID(100),HS(100),YR(100)
120  D$=CHR$(4)
130  FOR LO=1 TO 4: READ TI$(LO): NEXT
140  DATA 1. CREATE PANORAMA,2. SAVE PANORAMA,3. READ PANORAMA,4. QUIT
150  TEXT : HOME : HTAB 16: PRINT "OPTIONS"
160  VTAB 4: FOR LO=1 TO 4: HTAB 6: PRINT TI$(LO): NEXT LO
170  PRINT : HTAB 10: INPUT "WHICH OPTION? ";OP
180  ON OP GOTO 200,2000,3000,4000
190  GOTO 150
200  HGR : HCOLOR=3
205  XR=0:IB=0
210  HOME : VTAB 21
220  PRINT "1 DISPLAY A BUILDING BLOCK": PRINT "2 ERASE SCREEN"
230  PRINT "3 RETURN TO MAIN MENU"
240  HTAB 6: INPUT "WHICH OPTION? ";IZ
250  ON IZ GOTO 300,200,150
260  GOTO 210
300  REM *** DISPLAY A BUILDING BLOCK"
310  HOME : VTAB 21
320  INPUT "ID#, HORIZONTAL SPACE, AND VERTICAL POSITION: ";ID,HS,YR
325  IB=IB+1:ID(IB)=ID:HS(IB)=HS:YR(IB)=YR
330  XR=XR+HS
340  ON ID GOSUB 400,500,600,700,800
350  GOTO 210
400  REM *** DISPLAY A SMALL BIRD ***
410  HPLOT XR,YR-1: HPLOT XR+1,YR TO XR+2,YR: HPLOT XR+3,YR-1
420  XR=XR+4: RETURN
```

```
 500  REM *** DISPLAY A LARGE BIRD ***
 510  HPLOT XR,YR-2: HPLOT XR+1,YR-1: HPLOT XR+5,YR-2
 520  HPLOT XR+2,YR TO XR+3,YR: HPLOT XR+4,YR-1
 530  XR=XR+6: RETURN
 600  REM *** DISPLAY THE LEFT SIDE OF AN ISLAND ***
 610  HPLOT XR,YR TO XR+2,YR-3 TO XR+7,YR-3
 620  XR=XR+8: RETURN
 700  REM *** DISPLAY THE RIGHT SIDE OF AN ISLAND ***
 710  HPLOT XR,YR-3 TO XR+5,YR-3 TO XR+7,YR
 720  XR=XR+8: RETURN
 800  REM *** DISPLAY A PALM TREE ***
 805  HPLOT XR,YR-3 TO XR+11,YR-3
 810  HPLOT XR+4,YR-4 TO XR+7,YR-4
 815  HPLOT XR+5,YR-5 TO XR+6,YR-5
 820  HPLOT XR+6,YR-6: HPLOT XR+6,YR-7
 825  HPLOT XR+2,YR-6: HPLOT XR+2,YR-7
 830  HPLOT XR+3,YR-8: HPLOT XR+5,YR-8
 835  HPLOT XR+9,YR-8: HPLOT XR+1,YR-9
 840  HPLOT XR+4,YR-9 TO XR+6,YR-9: HPLOT XR+8,YR-9
 845  HPLOT XR+2,YR-10 TO XR+3,YR-10
 850  HPLOT XR+6,YR-10 TO XR+7,YR-10
 855  HPLOT XR+10,YR-10: HPLOT XR+4,YR-11 TO XR+6,YR-11
 860  HPLOT XR+9,YR-11: HPLOT XR+7,YR-12 TO XR+8,YR-12
 870  XR=XR+12: RETURN
 880  HPLOT XR+4,YR-9 TO XR+6,YR-9: HPLOT XR+8,YR-9
 890  HPLOT XR+2,YR-10 TO XR+3,YR-10
2000  REM *** SAVE A PANORAMA ***
2010  INPUT "NAME OF FILE TO BE SAVED: ";TI$
2020  INPUT "TYPE B TO BEGIN OR A TO APPEND: ";A$
2030  IF LEFT$(A$,1)="B" THEN 2100
2040  IF LEFT$(A$,1)="A" THEN 2200
2050  GOTO 2020
2100  REM *** BEGIN A NEW FILE ***
2110  PRINT D$;"OPEN";TI$: PRINT D$;"DELETE";TI$
2120  PRINT D$;"OPEN ";TI$
2130  PRINT D$;"WRITE ";TI$
2140  PRINT IB
2150  FOR LO=1 TO IB
2160  PRINT ID(LO): PRINT HS(LO): PRINT YR(LO):
2170  NEXT LO
2180  PRINT D$;"CLOSE ";TI$
2200  REM *** APPEND TO AN OLD FILE ***
2210  PRINT D$;"APPEND ";TI$
2220  PRINT D$;"WRITE ";TI$
2230  PRINT IB
2240  FOR LO=1 TO IB
2250  PRINT ID(LO): PRINT HS(LO): PRINT YR(LO)
2260  NEXT LO
2270  PRINT D$;"CLOSE ";TI$: GOTO 2300
2300  PRINT "FILE HAS BEEN SAVED": FOR LO=1 TO 1000: NEXT LO
```

```
2310   IB=0: GOTO 150
3000   REM *** READ A PANORAMA ***
3010   INPUT "NAME OF FILE TO BE READ: ";TI$
3020   XR=0
3030   HGR : HCOLOR=3: ONERR  GOTO 3500
3040   PRINT D$;"OPEN ";TI$
3050   PRINT D$;"READ";TI$
3060   INPUT IB
3070   IF IB=0 THEN 3100
3080   FOR RE=1 TO IB: INPUT ID,HS,YR:XR=XR+HS
3090   ON ID GOSUB 400,500,600,700,800
3100   NEXT RE
3110   FOR DE=1 TO 1000: NEXT DE
3150   HGR :XR=0: GOTO 3060
3500   FOR DE=1 TO 1000: NEXT DE: REM *** PROVIDE A DELAY ***
3510   POKE 216,0: REM *** TURN OFF ONERR GOTO ***
3520   PRINT D$;"CLOSE ";TI$
3530   GOTO 150
4000   END
```

3.3 BUILDING THE BUILDING BLOCKS

The program DESIGN1 lets you use the screen to design a building block very quickly. The building block "grows" on the screen while the viewer designs it. It is displayed alongside a tenfold magnification in an array of squares, (Figures 3.6A, 3.6B). The program first prompts for the number of rows and columns of pixels in the building block that is being designed, that is, the raster size. It then displays the appropriate array of squares with a dot in the square at the top left corner. This dot acts as a cursor. Each of the squares consists of 10×10 pixels. The program allows rasters that contain up to 300 squares; for example, 15×20 and 20×15 rasters are within its range. This limit can be increased by a very simple change.

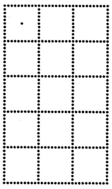


Figure 3.6A

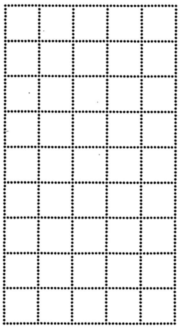


Figure 3.6B

Commands are given to the program by pressing the following keys:

→	Move the cursor one place to the right, or to the start of the next row if it is at the end of a row.
←	Move the cursor one place to the left, or to the end of the previous row if it is at the start of a row.

RETURN	Move the cursor down one place, or to the top of the column if it is in the bottom row.
U	Move the cursor up one place, or to the bottom of the column if it is in the top row.
P	Include the raster position of the cursor in the building block and move the cursor in the way described for →.
E	Erase the raster position of the cursor from the building block, and move the cursor in the way that was described for →.
H	Move the cursor to the top left position.
D	Display the list of raster coordinates of the building block that has been designed.
C	Clear the screen and allow another building block to be designed on the same raster.
ESCAPE	Terminate execution.

Whenever a P command is executed, the square that it addresses lights up, and so does the corresponding pixel in a display that is alongside the array of squares.

After requesting a 5×3 raster, and pressing the keys

P→ P→ P→ → P→ P→ P→ P

in succession, the screen appears as in Figure 3.7. The same display results from pressing the keys

P RETURN P U P RETURN → P → P RETURN P U P

The cursor appears as a bright dot on a square that has not been included in the building block, and as a dark dot on one that has.

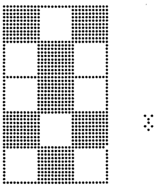


Figure 3.7

When a building block has been designed, its raster coordinates can be written down and used to construct programs similar to ALOHA1 and ALOHA2 that use the new building blocks. The program DESIGN2, in the next section, goes one stage further. It allows the viewer to design the building blocks interactively, and then to use them, without having to write any more programs or change any instructions.

The main variables in DESIGN1 are

XR	The number of columns in the raster
YR	The number of rows in the raster
XA, YA	The raster coordinates of the current cursor position
XO, YO	The raster coordinates of the previous cursor position
KP	The ASCII code of the current command

MA%	The array that shows which raster positions are lighted up. Successive elements of the array correspond to successive elements of the raster, proceeding through all the elements in one row before going on to the next. In the array, a 1 means that the element is lit up, and 0 means that it is not.

The program begins by prompting for XR and YR. The array of squares is displayed by statements 250–320. Line 440 delays until a key is pressed and then sets KP to its ASCII value. A branch is then made from one of the lines 450–540, if the key corresponded to one of the commands listed earlier, to further statements that perform the different actions that have been described.

Suggested variations

1. Allow the user to move the cursor dot in the NW, SW, SE, and NE directions by pressing further keys.
2. Allow larger rasters by using smaller squares and increasing the size of the MA% array.
3. Omit the array of squares and just use the cursor to identify actual pixels on the entire screen.

The DESIGN 1 program

```
100  REM *** DESIGN1 FOR BUILDING BLOCKS ***
110  TEXT : HOME : VTAB 21
120  INPUT "NUMBER OF COLUMNS AND ROWS? ";XR,YR
130  DIM MA(XR,YR)
250  REM *** DRAW THE GRID ***
260  HGR : HCOLOR=3
270  FOR Y=10 TO 10*(YR+1) STEP 10
280  HPLOT 10,Y TO 10*(XR+1),Y:
290  NEXT Y
300  FOR X=10 TO 10*(XR+1) STEP 10
310  HPLOT X,10 TO X,10*(YR+1)
320  NEXT X
400  REM *** SET CURSOR AT TOP LEFT HAND POSITION ***
410  XA=1:YA=1:SW=0: GOSUB 700
430  REM *** AWAIT A KEYPUSH ***
440  GET A$:KP=128+ASC(A$):XO=XA:YO=YA
450  IF KP=136 THEN XA=XA-1: GOTO 630: REM *** LEFT ARROW ***
460  IF KP=149 THEN 610: REM *** RIGHT ARROW ***
470  IF KP=141 THEN YA=YA+1: GOTO 630: REM *** RETURN ***
480  IF KP=213 THEN YA=YA-1: GOTO 630: REM *** U ***
490  IF KP=200 THEN  GOSUB 750: GOTO 410: REM *** H ***
500  IF KP=208 THEN MA(XA,YA)=1: GOSUB 800: GOTO 610: REM *** P ***
510  IF KP=197 THEN MA(XA,YA)=0: HCOLOR=0: GOSUB 800: HCOLOR=3: GOTO 610:  REM ***
      E ***
520  IF KP=195 THEN  CLEAR : GOTO 110: REM *** C ***
530  IF KP=196 THEN  GOSUB 900: REM *** D ***
```

```
540  IF KP=155 THEN  STOP : REM *** ESC ***
590  GOTO 440
600  REM *** MOVE CURSOR FORWARD ***
610  XA=XA+1
620  REM *** CORRECT FOR MOVEMENT OUT OF ROW OR COLUMN ***
630  IF XA<1 THEN YA=YA-1:XA=XR: GOTO 650
640  IF XA>XR THEN XA=1:YA=YA+1
650  IF YA<1 THEN YA=YR: GOTO 630
660  IF YA>YR THEN YA=1: GOTO 630
670  GOSUB 700: GOTO 430
700  REM *** PLOT THE CURSOR ***
710  IF MA(XA,YA) THEN  HCOLOR=0
720  HPLOT 10*XA+5,10*YA+5
730  HCOLOR=3
740  REM *** TEST IF DESIGN WAS JUST STARTED ***
750  IF SW=0 THEN SW=1: RETURN
760  REM *** ERASE THE PREVIOUS CURSOR ***
770  HCOLOR=3*(MA(X0,Y0)=1)
780  HPLOT 10*X0+5,10*Y0+5
790  HCOLOR=3: RETURN
800  REM *** PLOT AN ELEMENT ***
810  FOR X=10*XA+1 TO 10*XA+9
820  HPLOT X,10*YA+1 TO X,10*YA+9
830  NEXT X
840  HPLOT 10*XR+XA+20,YA+40
850  RETURN
900  REM *** DISPLAY COORDINATES ***
910  TEXT : HOME
920  FOR Y=1 TO YR: FOR X=1 TO XR
930  IF MA(X,Y) THEN  PRINT X;",";Y
940  NEXT X,Y
950  PRINT : PRINT "PRESS ANY KEY TO CONTINUE";: GET A$: PRINT
960  REM *** BRING HI-RES SCREEN 1 ONTO DISPLAY ***
970  POKE -16304,0: POKE -16297,0
980  RETURN
```

3.4 *SOUTH PACIFIC* REVISITED

The program DESIGN2 begins by prompting the user to type one of the following options:

1. To design a building block (which can be stored temporarily with an ID number from 1 to 20 for the duration of the current run)
2. To construct a panorama
3. To display the data that represent, within the program, a building block that has been designed and stored temporarily, so that it may be written down for reuse in later runs by means of option 4
4. To allow the user to type in the data that represent a building block that has been written down previously
5. To quit

When the user types 1, the program behaves like DESIGN1, except that pressing the ESCAPE key brings back the main menu, and pressing the S key makes the program display the message

```
TYPE S TO STORE OR C TO CANCEL:
```

Typing C (or any other key except S) allows the building block to be edited further. Typing S makes the program display

```
NEW ID# (DEFAULT=1):
```

if no building blocks have been stored yet. The message is

```
NEW ID# (DEFAULT=2):
```

after one building block has been stored, and so on. Pressing the RETURN key without typing a number first makes the program use the default value. Typing an ID number that has already been used destroys the building block with that number, and any building blocks with higher numbers. After a building block has been stored by typing an ID number in response to this request, the program goes back to give the initial prompt again.

To construct the ALOHA panoramas, you first construct the five building blocks shown in Figure 3.3, using the tactics described for DESIGN1. The small bird is designed on a 4×2 raster, and stored as building block 1. The large bird, the left and right sides of the island, and the tree-bearing portion are designed on 6×3, 8×4, 8×4, and 12×14 rasters, respectively, and stored as building blocks 2–5.

To display a panorama, the user types 2, after the building blocks have been constructed and stored. The program then goes through the same prompting sequence as ALOHA1, and the user responds in the way that was described. For example, the panorama of Figure 3.1 is produced by the responses given in Section 2.4.

You type 3 to display the internal representation of a building block. This representation illustrates a commonly used tactic of data structure work. The program then prompts for the ID number. When you have typed this, the program displays the following items (to be explained):

`NC%(ID), NR%(ID), NP%(ID), I0%(ID)`
`XP%(IQ), YP%(IQ)` for `IQ=I0%(ID)+1` to `I0%(ID)+NP%(ID)`

To type in such data, you request option 4 in response to the initial prompt. The program then prompts in turn for the items just specified.

The program DESIGN2 uses all the variables and arrays of DESIGN1, and several more as well. The way these are used can be explained by the *South Pacific* example. Building block 1 is the small bird in Figure 3.8A. The program represents this by the following items:

`NC%(1)`	The number of columns in the raster that forms it (4)
`NR%(1)`	The number of rows in the raster that forms it (2)
`NP%(1)`	The number of pixels that are lighted up to form it (4)
`XP%(1)`	The x (column) value of the first of these points, in the 4×2 raster (1)
`YP%(1)`	The y (row) value of the first of these points, in the 4×2 raster (1)

`XP%(2)`	The corresponding x value of the second of these points (2)
`YP%(2)`	The corresponding y value of the second of these points (2)
`XP%(3)`	The corresponding x value of the third of these points (3)
`YP%(3)`	The corresponding y value of the third of these points (2)
`XP%(4)`	The corresponding x value of the fourth of these points (4)
`YP%(4)`	The corresponding y value of the fourth of these points (1)
`I0%(1)`	The number of entries that precede the data for the small bird in the XP% and YP% arrays (0)

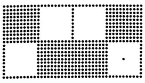


Figure 3.8A

Building block 2 is the large bird in Figure 3.8B.The program represents this by the following items:

```
NC%(2)=6, NR%(2)=3, NP%(2) =6, I0%(2)=4
XP%(5)=1, YP%(5)=1, XP%(6) =2, YP%(6)=2
XP%(7)=3, YP%(7)=3, XP%(8) =4, YP%(8)=3
XP%(9)=5, YP%(9)=2, XP%(10)=6, YP%(10)=1
```

The NC%, NR%, NP%, and I0% values show that it is defined on a 6×3 raster, 6 pixels are lighted up, and four items precede it in the XP% and YP% arrays.

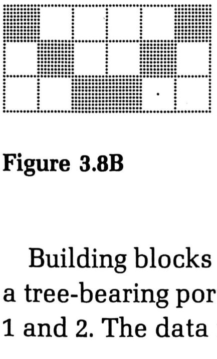


Figure 3.8B

Building blocks 3–5 are, respectively, the left and right sides of the island and a tree-bearing portion. The program represents these in the same way as blocks 1 and 2. The data in the first four arrays that define the five building blocks are

ID:	1,	2,	3,	4,	5
NC%:	4,	6,	8,	8,	12
NR%:	2,	3,	4,	4,	14
NP%:	4,	6,	11,	11,	45
I0%:	0,	4,	10,	21,	32

This table shows that the coordinates of the first block are stored in XP%(1), YP%(1) to XP%(4), YP%(4), and the coordinates of the second block in XP%(5), YP%(5) to XP%(10), YP%(10), as explained already. The coordinates of the third block are stored in XP%(11), YP%(11) to XP%(21), YP%(21). The coordinates of the fourth block are stored in XP%(22), YP%(22) to XP%(32), YP%(32), and those of the fifth block in XP%(33), YP%(33) to XP%(77), YP%(77).

The main variables and arrays used in DESIGN2 thus are as follows:

XR	Number of columns of pixels in a building block that is being designed
YR	Number of rows of pixels in a building block that is being designed
MA%(500)	Flags (0 or 1) to show whether a pixel is lighted up, in a building block that is being designed (see DESIGN1)
NC%(20)	Number of columns of pixels in a building block that has been stored
NR%(20)	Number of rows of pixels in a building block that has been stored
NP%(20)	Number of pixels that light up in a building block that has been stored
I0%(20)	Number of elements in the XP%, YP% arrays before the elements that define a building block
IN	Index of elements that are stored in XP%, YP% in the cycle that stores a building block
IQ	Index of elements in XP% and YP% in the cycle that displays a building block as part of a panorama
ID	Identification number of a building block, in the portion of the program that stores it
DI	Identification number of a building block, in the portion of the program that displays it as part of a panorama
IP	Takes successive values 1 to NP%(DI) for the successive elements of XP%, YP% that represent the DIth building block, when this is displayed as part of a panorama
XX	x coordinate of a point in the raster for a building block
YY	y coordinate of a point in the raster for a building block

Statements that are common to DESIGN1 and DESIGN2 have the same statement numbers. The initial prompt of DESIGN2 has more options. These are displayed by lines 160–190. The portions of DESIGN2 to draw a grid, to design a building block and to display the coordinates are the same as in DESIGN1, from the beginning at line 250 through line 980 with the addition of line 550 and a change in line 930. After the statements that prompt the user for the ID to store a building block, N0 is set to 0 in line 1060 for use as an index that takes successive values 1, 2, ... for successive elements in the array MA%. The values of I0%(ID), NR%(ID), and NC%(ID) are set by the further statements in this line. The nested FOR loops in lines 1070–1090 test the elements of MA% that correspond to successive x, y positions within the raster (denoted XX, YY). Whenever a 1 is found, IN is increased by 1 and XP%(IN), YP%(IN) are set to the appropriate x, y values by line 1080. After the loop, line 1100 sets NP%(ID) to the number of points whose coordinates were transferred to the XP%, YP% arrays and increases ID by 1 for the next building block.

The portion that displays a panorama starts at statement 2000. It prompts and reaches statement 2090 when a building block is to be displayed. HP, the horizontal starting position, is increased by HS, the value provided by the user in response to the prompt. The variable IQ, which is used as the index that addresses the elements of XP%, YP% for the DIth building block, is set to I0%(DI) by line 2100. Then, in the FOR loop in lines 2110–2130, it is increased by 1 for each of the NP%(DI) points in the building block. These are plotted at the appropriate positions by line 2120. Following the loop, the horizontal position HP is increased by NC%(DI), the width of the building block, and the program branches back to prompt the user again.

Suggested variations

1. Make the changes and provide the further options (3–5) suggested for ALOHA1.
2. Allow the panoramas to be saved in text files and redisplayed.
3. Make the changes and provide the further options suggested for DESIGN1.
4. Allow the building blocks to be saved in text files and read for subsequent use.

The DESIGN2 program

```
100  REM *** DESIGN2 FOR BUILDING BLOCKS AND PANORAMAS ***
110  LOMEM: 16384
120  DIM MA%(300),XP%(500),YP%(500)
130  DIM NR%(20),NC%(20),NP%(20),I0%(20)
140  IN=0:ID=1
145  REM *** DISPLAY MAIN MENU ***
150  TEXT : HOME
160  PRINT "OPTIONS": PRINT : PRINT "1 TO DESIGN A BUILDING BLOCK"
170  PRINT "2 TO CONSTRUCT A PANORAMA": PRINT "3 TO DISPLAY DATA"
180  PRINT "4 TO TYPE IN A BUILDING BLOCK": PRINT "5 TO QUIT"
190  INPUT "WHICH OPTION? ";ZI
200  ON ZI GOTO 220,2000,3000,4000,5000
210  HOME : GOTO 140
215  REM *** CONSTRUCT A BUILDING BLOCK ***
220  INPUT "NUMBER OF COLUMNS AND ROWS? ";XR,YR
230  N=1: FOR LO=1 TO 300:MA%(LO)=0: NEXT
250  REM *** DRAW THE GRID ***
260  HGR : HCOLOR=3
270  FOR Y=10 TO 10*(YR+1) STEP 10
280  HPLOT 10,Y TO 10*(XR+1),Y
290  NEXT Y
300  FOR X=10 TO 10*(XR+1) STEP 10
310  HPLOT X,10 TO X,10*(YR+1)
320  NEXT X
400  REM *** SET CURSOR AT TOP LEFT HAND POSITION ***
410  XA=1:YA=1:SW=0: GOSUB 700
430  REM *** AWAIT A KEYPUSH ***
440  GET A$:KP=128+ASC(A$):X0=XA:Y0=YA
450  IF KP=136 THEN XA=XA-1: GOTO 630: REM *** LEFT ARROW ***
460  IF KP=149 THEN 610: REM *** RIGHT ARROW ***
470  IF KP=141 THEN YA=YA+1: GOTO 630:REM *** RETURN ***
480  IF KP=213 THEN YA=YA-1: GOTO 630: REM *** U ***
```

```
 490  IF KP=200 THEN  GOSUB 750: GOTO 410: REM *** H ***
 500  IF KP=208 THEN  GOSUB 5500:MA%(NO)=1: GOSUB 800: GOTO 610:REM *** P ***
 510  IF KP=197 THEN  HCOLOR=0: GOSUB 5500:MA%(NO)=0: GOSUB 800: HCOLOR=3: REM *E*
 520  IF KP=195 THEN 230: REM *** C ***
 530  IF KP=196 THEN  GOSUB 900: REM *** D ***
 540  IF KP=155 THEN 150: REM *** ESC ***
 550  IF KP=211 THEN  GOSUB 1000: REM *** S ***
 590  GOTO 430
 600  REM *** MOVE CURSOR FORWARD ***
 610  XA=XA+1
 620  REM *** CORRECT FOR MOVEMENT OUT OF ROW OR COLUMN ***
 630  IF XA<1 THEN YA=YA-1:XA=XR: GOTO 650
 640  IF XA>XR THEN XA=1:YA=YA+1
 650  IF YA<1 THEN YA=YR: GOTO 630
 660  IF YA>YR THEN YA=1: GOTO 630
 670  GOSUB 700: GOTO 430
 700  REM *** PLOT THE CURSOR ***
 710  GOSUB 5500: IF MA%(NO) THEN  HCOLOR=0
 720  HPLOT 10*XA+5,10*YA+5
 730  HCOLOR=3
 740  REM *** ERASE THE PREVIOUS CURSOR IF NECESSARY ***
 750  IF SW=0 THEN SW=1: RETURN
 770  HCOLOR=3*(XO+XR*(YO-1)=1)
 780  HPLOT 10*XO+5,10*YO+5
 790  HCOLOR=3: RETURN
 800  REM *** PLOT AN ELEMENT ***
 810  FOR X=10*XA+1 TO 10*XA+9
 820  HPLOT X,10*YA+1 TO X,10*YA+9
 830  NEXT X
 840  HPLOT 10*XR+XA+20,YA+40
 845  SW=0
 850  RETURN
 900  REM *** DISPLAY COORDINATES ***
 910  TEXT : HOME
 920  FOR YA=1 TO YR: FOR XA=1 TO XR
 930  GOSUB 5500: IF MA%(NO) THEN  PRINT XA;",";YA
 940  NEXT XA,YA
 950  PRINT : PRINT "PRESS ANY KEY TO CONTINUE";: GET A$: PRINT
 960  REM *** BRING HI-RES SCREEN 1 ONTO DISPLAY ***
 970  POKE -16304,0: POKE -16297,0
 980  RETURN
1000  REM *** STORE THE COORDINATES FOR LATER USE ***
1010  HOME : VTAB 21
1020  PRINT "TYPE S TO STORE OR C TO CANCEL: ";: GET A$
1030  IF A$<>"S" THEN  HOME : VTAB 21: GOTO 440
1040  PRINT "NEW ID# (DEFAULT=";ID;"): ";: INPUT ID$
1050  IF LEN(ID$)>0 THEN ID=VAL(ID$)
1060  NO=0:IO%(ID)=IN:NR%(ID)=YR:NC%(ID)=XR
```

```
1070  FOR YY=1 TO YR: FOR XX=1 TO XR
1080  NO=NO+1: IF MA%(NO) THEN IN=IN+1:XP%(IN)=XX:YP%(IN)=YY
1090  NEXT XX,YY
1100  NP%(ID)=IN-IO%(ID):ID=ID+1
1110  HOME : VTAB 21: RETURN
2000  REM *** CONSTRUCT A PANORAMA ***
2010  HP=0: HGR : HCOLOR=3
2020  REM *** DISPLAY SECONDARY MENU ***
2030  HOME : VTAB 21
2040  PRINT "TYPE 1 TO DISPLAY A BUILDING BLOCK"
2050  PRINT "TYPE 2 TO CLEAR THE SCREEN"
2060  INPUT "TYPE 3 TO RETURN TO MAIN MENU: ";Z
2070  ON Z GOTO 2090,2010,150
2080  GOTO 2030
2090  INPUT "ID#, HORIZONTAL SPACING, VERTICAL POSITION";DI,HS,VP
2100  HP=HP+HS:IQ=IO%(DI)
2110  FOR IP=1 TO NP%(DI)
2120  IQ=IQ+1: HPLOT HP+XP%(IQ),VP+YP%(IQ)
2130  NEXT IP:HP=HP+NC%(DI): GOTO 2030
3000  REM *** DISPLAY CODED DATA FOR ANY BUILDING BLOCK ***
3010  REM *** STATEMENTS DO THIS FOR THE CURRENT BLOCK ***
3020  TEXT : HOME :BN=ID: INPUT "ID#? ";ID
3025  PRINT "NUMBER OF COLUMNS IS ";NC%(ID)
3030  PRINT "NUMBER OF ROWS IS ";NR%(ID)
3035  PRINT "NUMBER OF POINTS IS ";NP%(ID)
3037  PRINT "BASE ADDRESS IN ARRAY IS ";IO%(ID)
3040  PRINT "X","Y"
3050  POKE 34,5: REM *** LOWERS TOP OF SCROLLING WINDOW ***
3060  FOR L=IO%(ID)+1 TO IO%(ID)+NP%(ID)
3070  PRINT XP%(L),YP%(L)
3080  NEXT L
3090  PRINT : PRINT "PRESS ANY KEY TO CONTINUE";
3100  GET A$:ID=BN: POKE 34,0: GOTO 150
4000  REM *** INPUT CODED BUILDING BLOCK ***
4010  ID=ID+1:IO%(ID)=IN
4020  PRINT "ID#(DEFAULT=";ID;"): ";: INPUT ID
4030  INPUT "NUMBER OF COLUMNS AND ROWS? ";NC%(ID),NR%(ID)
4040  INPUT "NUMBER OF POINTS? ";NP%(ID)
4050  IN=IO%(ID)
4060  FOR L=1 TO NP%(ID)
4070  IN=IN+1: INPUT "X AND Y VALUES? ";XP%(IN),YP%(IN)
4080  NEXT L
4090  PRINT "BUILDING BLOCK HAS BEEN STORED"
4100  FOR DE=1 TO 1000: NEXT DE
4110  GOTO 150
5000  REM *** END ***
5010  END
5500  NO=XA+XR*(YA-1): RETURN
```

3.5 USING APPLE SHAPE TABLES

You can carry the building block principle still further on the Apple. This has special commands that let you

1. Create "shape tables" that contain coded "shape definitions" and
2. Display the shapes that these specify, wherever you want them on the screen, and with whatever magnification and rotation that is needed

The program STATIC1 uses this feature to let you produce patterns like those in Figures 3.9A–3.9F.

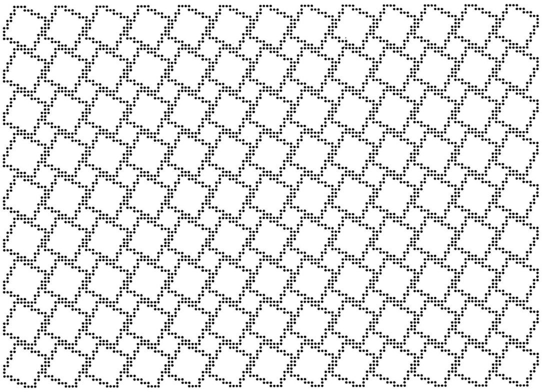


Figure 3.9A

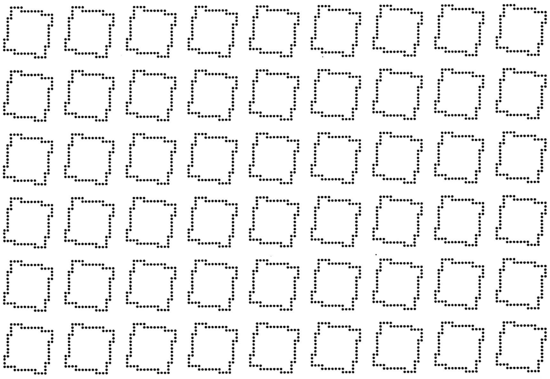


Figure 3.9B

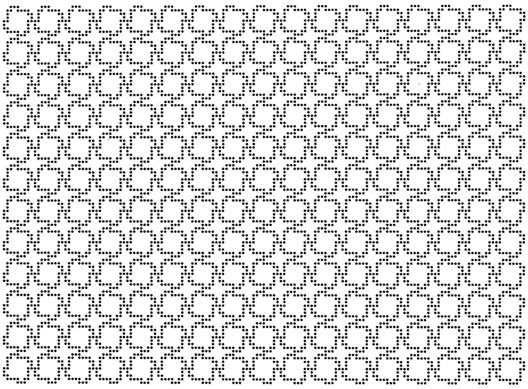


Figure 3.9C

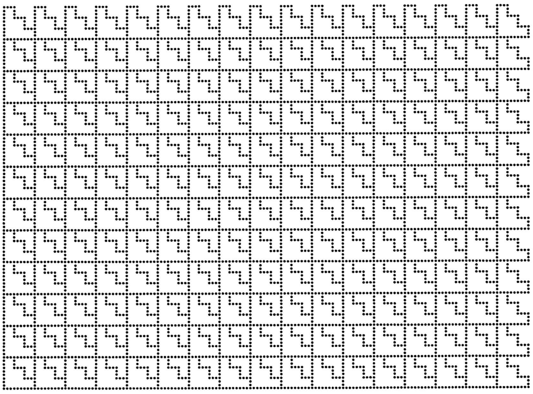


Figure 3.9D

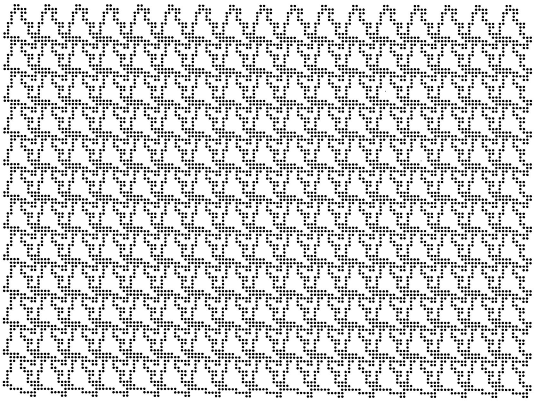


Figure 3.9E

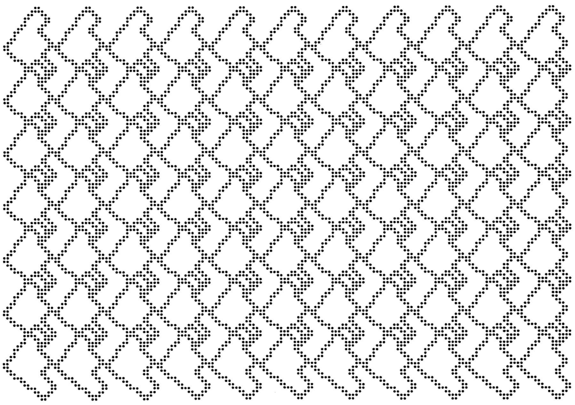


Figure 3.9F

The patterns in Figures 3.9A–3.9C are formed from a single square that is stored as shape 1 in the shape table (left side of Figure 3.10), and the patterns in Figures 3.9D–3.9F are formed from a stepped outline that is stored as shape 2 (right side of Figure 3.10).

Figure 3.10

The program prompts for (1) a shape number, that is, 1 or 2, (2) the magnification, which can be between 1 and 255, and (3) the angle of rotation, in units of 5.625° which is 1/16 of a right angle—for example, specifying 3 for the rotation tilts the shape by 16.875°. The program then repeats the shape across the screen in the vertical and horizontal directions, after scaling and rotating it by the required amounts.

The main variables in the program are

`NU`	The shape number
`SI`	The scaling factor
`RT`	The rotation (in units of 90/16=5.625°)
`X,Y`	The x, y values of the plotting position
`C`	A variable that is used to store the codes in the shape tables

The first part of the program, lines 120–140, creates the shape table by reading a series of codes from a DATA statement and poking these into consecutive addresses, beginning at 7676. (The principle that was used to construct the codes that represent the shapes will be explained soon.) A shape table may contain the definitions of many shapes. The program then prompts for the shape number, the size (that is, scale factor), and the rotation. The patterns in the illustration were produced by the values (1,4,4), (1,4,2), (1,3,4), (2,4,0), (2,4,4), and (2,5,4), respectively. Line 200 sets the scale and rotation with the instructions

```
SCALE=SI: ROT=RT
```

The remaining statements draw the shape at position X, Y by means of the instruction

```
DRAW NU AT X,Y
```

using values computed in a nested FOR loop.

The program prompts

```
PRESS ANY KEY TO CONTINUE
```

When you have pressed a key, the program repeats the process that starts with the prompt for a shape number, size, and rotation.

The shape table for STATIC1 was designed by drawing the square and the stepped outline on graph paper, in 3×3 and 4×4 rasters, respectively, and traversing the pixels that form the shapes as shown in Figures 3.11A and 3.11B.

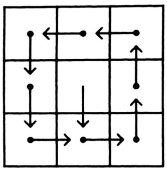


Figure 3.11A

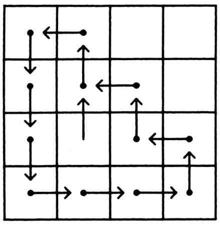


Figure 3.11B

These can be "unraveled" to give the lists of arrows shown in Figure 3.12 (a dot at the tail means that the pixel where it starts is to be lighted up).

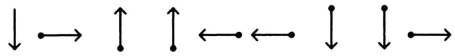


Figure 3.12A

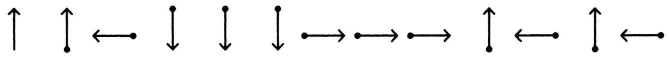


Figure 3.12B

In shape diagrams such as these, an arrow can designate a move that is either up, right, down, or left from an unlighted raster position, or up, right, down, or left from a position that is to be lighted up. These eight alternatives are represented by the binary numbers 000, 001, 010, 011, 100, 101, 110, and 111, respectively.

The sequence of binary numbers is packed into a succession of 8 bit words, going from right to left in each word. The two leftmost bits are set to 0, and are not used for a code unless it is 001, 010 or 011. It is stored as 01, 10, or 11, respectively, in these three cases. The code 000 is ignored if there are only zeros to its left, and this must be taken into account when packing a shape table. A final word containing eight zeros, 00 000 000, is put at the end of a shape definition. The square thus is coded in eight words. The table that follows below shows what is in these words:

1. In binary, with spaces between the three groups of bits, to emphasize the relation to the diagram
2. In binary with a space between the left four bits and the right four bits
3. In hexadecimal
4. As the equivalent decimal number

Of course the decimal number (4) can be constructed directly from the first form (1), but the intermediate forms (2 and 3) make the conversion easier when you do it with pencil and paper. Here is the table.

Binary	Binary	Hexadecimal	Decimal
00 101 010	0010 1010	2A	42
00 100 100	0010 0100	24	36
00 111 111	0011 1111	3F	63
00 110 110	0011 0110	36	54
00 000 101	0000 0101	05	05
00 000 000	0000 0000	00	00

The corresponding encoding of the second shape is

Binary	Binary	Hexadecimal	Decimal
00 100 000	0010 0000	20	32
00 110 111	0011 0111	37	55
00 110 110	0011 0110	36	54
00 101 101	0010 1101	2D	45
00 100 101	0010 0101	25	37
00 100 111	0010 0111	27	39
00 000 111	0000 0111	07	07
00 000 000	0000 0000	00	00

A complete shape table that contains n shape definitions ($n=2$ in the present example) consists of $2n+2$ numbers (to be explained shortly), followed by the numbers that define the individual shapes, that is 42, 36, ..., 5, 0 for shape 1 and 32, 55, ..., 7, 0 for shape 2.

The first two numbers in the shape table show how many shapes it contains. These numbers are, respectively, the remainder and the quotient when n is divided by 256—2 and 0 in the present example. The next pair specify in the same way just how many numbers precede the definition of shape 1. These are accordingly the remainder and quotient when 6 is divided by 256—6 and 0 in the present example. The next two numbers are 12 and 0 to specify, in the same way, just how many numbers precede the definition of shape 2.

The complete shape table thus is

2, 0, 6, 0, 12, 0, 42, 36, 63, 54, 5, 0, 32, 55, 54, 45, 37, 39, 7, 0

The shape table is stored in successive words starting at address 7676 by the present program. This is safe in usual circumstances. Deciding where to put it when a program is very large or when other special conditions prevail requires an understanding of the storage map, which is discussed in the references cited at the beginning of Chapter 8.

Suggested variations

1. Construct different patterns with the shapes in STATIC1.
2. Construct some different shapes and alter STATIC1 to use these. (*Hint:* change the DATA statement and the limit on the FOR loop that reads it.)
3. Write a program to prompt for the successive numbers in a hand-coded shape table and to store these as a text file.
4. Alter ALOHA1 and ALOHA2 to construct scenes that consist of shapes that have been defined in shape tables and that are selected and arranged by responses to simple prompts.

5. Extend this to allow the shapes to be rotated and scaled by responses to further prompts.
6. Use the interactive design tactics of Section 3.3 and 3.4 to construct a shape table mechanically.
7. Combine the features of suggestions 5 and 6.

The STATIC1 program

```
100  REM *** STATIC1-USES TWO SHAPES ***
110  REM *** STORE THE SHAPE TABLE ***
120  FOR LO=7676 TO 7695: READ C: POKE LO,C: NEXT
130  DATA 2,0,6,0,12,0,42,36,63,54,5,0,32,55,54,45,37,39,7,0
140  POKE 232,252: POKE 233,29: REM *** STORE BASE ADDRESS (7676=29*256+252) ***
150  HGR : HCOLOR=3
160  HOME : VTAB 21
170  INPUT "WHICH SHAPE? ";NU
180  INPUT "SIZE AND ROTATION? ";SI,RT
190  IF SI>20 THEN SI=20
200  SCALE=SI: ROT=RT
210  FOR Y=20 TO 120 STEP (SI*3)
220  FOR X=50 TO 200 STEP (SI*3)
230  DRAW NU AT X,Y
240  NEXT X,Y
250  PRINT "PRESS ANY KEY TO CONTINUE";: GET A$: PRINT : GOTO 150
```

3.6 SNOWFLAKES AND DIATOMS

The program STATIC2 uses the shape commands to create a pattern that has radial symmetry (Figure 3.13). Any line through the center of the pattern divides it into two halves that are superimposed exactly by rotating either of them by 180°. Snowflakes, diatoms, and many other naturally occurring objects have this symmetry.

The program uses a single shape, in fact the simple 3×3 square, that "begins" at its center point, as shown in Figure 3.11. This is drawn eight times on each of 16 lines that radiate at uniform angular intervals from the center of the screen. The centers of the eight squares on each radial line are spaced uniformly, and the interval is the same for each line. The centers of the squares thus lie on eight concentric circles. The rotation of the squares about their centers, and the scale, increase uniformly from the center of the pattern by amounts that you specify in response to a prompt.

The screen coordinates x, y of the center of each square are found from simple trigonometry:

$$x=140+r\cos\theta, \qquad y=95+r\sin\theta$$

where r is the distance of the center of the square from the center of the screen and θ is the angle between the radial line and a horizontal line through the center of the screen. The values of $\sin\theta$ and $\cos\theta$ are calculated at the start of the program for the 16 values that are needed, and stored in an array, to reduce the time needed in later computations.

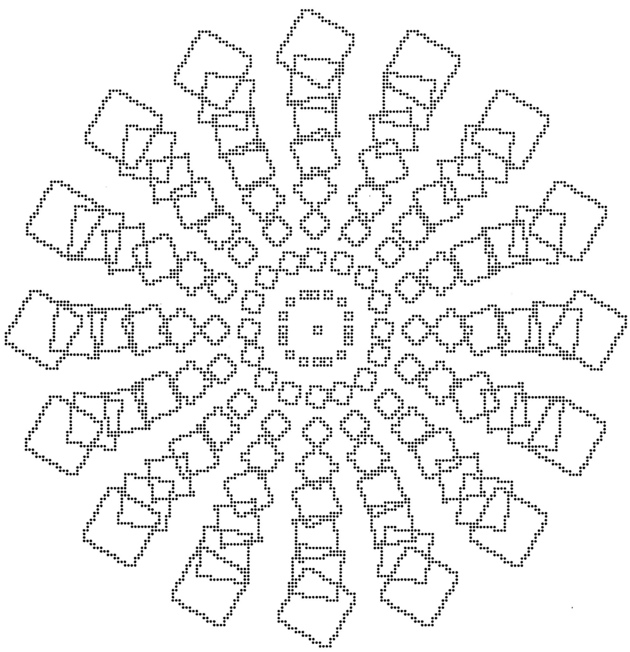


Figure 3.13

The main variables in the program are

TV(1,N)	The cosine of $(N\times 2\pi)/16$, used to compute the x value of the center of a square on the Nth radial line
TV(2,N)	The sine of $(N\times 2\pi/16$, used to compute the corresponding y value
RT	The increase in the rotation of the square about its center, for successive squares along a radial line
RO	The actual rotation
ST	The increase in the scale of the square, for successive squares along a radial line
SC	The actual scale
FA	The distance of the center of a square from the center of the screen
L1	Index of the FOR loop for successive distances from the center of the screen
L2	Index of the FOR loop for successive radial lines, and of the elements in TV

The program first computes and stores, in the array TV, the sines and cosines that are needed later. It uses a FOR loop that contains a READ statement to get the numbers for the shape table from a DATA statement, and pokes these into the appropriate memory locations. Then it prompts for the amounts by which the scale and rotation of the square should increase along the radial lines. The program then uses a nested FOR loop to draw the squares at appropriate positions on the screen, scaled and rotated by the necessary amounts. It then repeats the process from the prompt for scale and rotation factors.

Suggested variations

1. Change the shape that the program displays.
2. Let the user specify the number of "spokes."
3. Create patterns with radial symmetry using several shapes.
4. Imagine the screen divided into quarters and produce patterns with radial symmetry in each.

The STATIC2 program

```
100   REM *** STATIC2- USES SHAPE TABLES ***
110   REM *** STORE THE TABLE ***
120   FOR LO=7676 TO 7685: READ C: POKE LO,C: NEXT LO
130   DATA 1,0,4,0,58,36,45,54,7,0
140   POKE 232,252: POKE 233,29: REM *** STORE THE BASE ADDRESS OF THE TABLE ***
150   REM *** SET UP TABLE OF SINES AND COSINES ***
160   DIM TV(2,16)
170   FOR LO=1 TO 16:TV(1,LO)=COS(TH):TV(2,LO)=SIN(TH):TH=TH+.3925: NEXT LO
180   HOME : VTAB 21: REM *** PROMPT FOR INPUT ***
190   PRINT "PLEASE ENTER THE AMOUNTS BY WHICH THE "
200   PRINT "ROTATION AND SCALE WILL INCREASE, IN"
210   INPUT "THE RANGES 0 TO 15 AND 0 TO 4: ";RT,ST
220   RO=0:SC=1:FA=10: HGR : HCOLOR=3: SCALE=1: ROT=0
230   POKE -16302,0: REM *** SWITCH TO FULL SCREEN GRAPHICS ***
240   DRAW 1 AT 140,95
250   FOR L1=1 TO 8
260   FOR L2=1 TO 16
270   DRAW 1 AT 140+TV(1,L2)*FA,95+TV(2,L2)*FA
280   NEXT L2
290   FA=FA+10:SC=SC+ST: SCALE=SC:RO=RO+RT: ROT=RO
300   NEXT L1
310   FOR DE=1 TO 1000: NEXT DE: POKE -16301,0: REM *** MIXED SCREEN GRAPHICS ***
320   GOTO 180
```

3.7 SOME HAPPY SQUARES

The program SQUARES displays two patterns that fill the screen in rapid alternation (Figures 3.14A and 3.14B). It uses the "page" feature of the Apple (to be explained).

The first pattern is produced by applying shape commands to a simple square that is rotated about its center. The command

```
HGR2
```

leaves this pattern intact, in an area of memory called the first page of high-resolution graphics, and allows the second pattern to be drawn in another area of memory called the second page of high-resolution graphics. The second pattern is drawn by using the shape commands to rotate a square about a corner. The command

```
POKE -16300,0
```

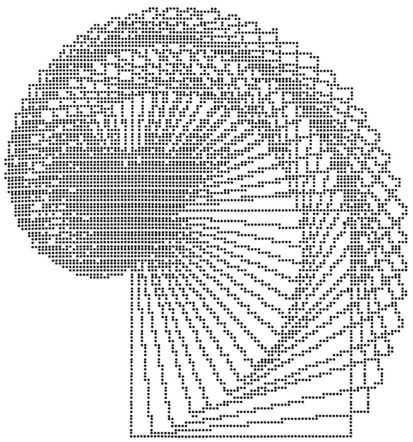


Figure 3.14A

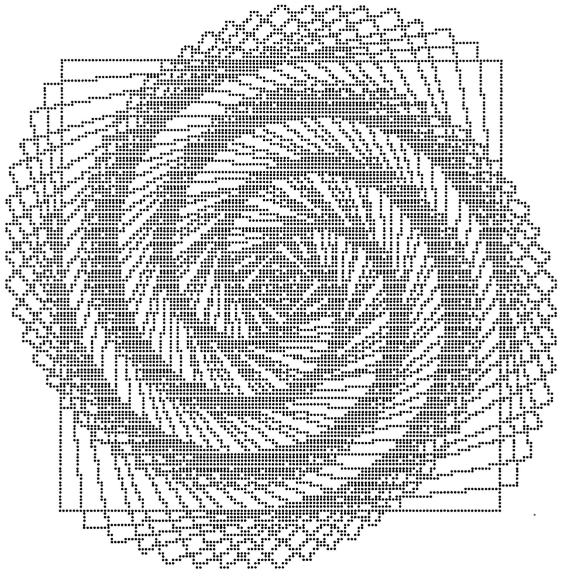


Figure 3.14B

brings the first pattern back to the screen, without destroying the second pattern, which remains in the second page of high-resolution graphics. The command

```
POKE -16299,0
```

reverses this switch.

The Apple actually uses six separate areas of memory to store information that is displayed almost instantaneously, when an appropriate command is given. Two of these pages are for text, two are for low-resolution graphics, and two are for high-resolution graphics. The commands TEXT, GR, HGR, and HGR2 clear and connect to the screen, respectively, the first page of text, the first page of low-resolution graphics, the first page of high-resolution graphics, and the second page of high-resolution graphics. Using the second pages of text and low-resolution graphics is relatively awkward.

The program first stores the shape table, using a FOR loop that reads the relevant numbers from a DATA statement, and pokes these into the required location. The table defines a square that "begins" at its center, and another square that "begins" at a corner. These two shapes are used to produce the two patterns by scaling and rotation about the beginning points, using SCALE and ROT commands in the FOR loops that are indexed by L in the HGR and HGR2 modes, respectively. The rest of the program displays the two patterns in alternation, using POKEs to switch pages, and a FOR loop to provide a delay.

Note that the commands HGR and HGR2 cannot be used to make the patterns alternate rapidly because HGR erases the first page and HGR2 erases the second page to let you draw something new in them.

Suggested variations

1. Let the user specify some parameters that define the shapes that appear in alternation.
2. Adapt ALOHA2 to display two panoramas in alternation.

The SQUARES program

```
100  REM *** SQUARES - USES A SHAPE TABLE ***
110  REM *** STORE THE TABLE ***
120  FOR LO=7676 TO 7690: READ C: POKE LO,C: NEXT
130  DATA 2,0,6,0,9,0,53,39,0,58,36,45,54,7,0
140  POKE 232,252: POKE 233,29: REM *** STORE THE BASE ADDRESS ***
150  HGR : HCOLOR=3: REM *** DRAW ON PAGE 1 ***
160  FOR L=1 TO 64: SCALE=L: ROT=L: DRAW 1 AT 140,80: NEXT L
170  HGR2 : HCOLOR=3: REM *** DRAW ON PAGE 2 ***
180  FOR L=1 TO 64: SCALE=L: ROT=L: DRAW 2 AT 140,80: NEXT
190  POKE -16300,0: FOR DE=1 TO 80: NEXT DE: REM *** DISPLAY PAGE 1 AND PAUSE ***
200  POKE -16299,0: FOR DE=1 TO 80: NEXT DE: REM *** DISPLAY PAGE 2 AND PAUSE ***
210  GOTO 190
```

3.8 THROWING THE HAMMER

The program HAMMER lets you use the paddles (if you have them on your Apple) to control the scale and rotation of a hammer on the screen (actually the outline of a hammer—a real one is not recommended). The hammer (Figure 3.15) is defined as a shape using the tactics already described. The statements

```
HGR: HCOLOR=3: HPLOT 0,0: CALL 62454
```

turn the screen completely white. In general the statement

```
CALL 62454
```

turns the screen to the color that was used to plot most recently. The statement

```
POKE -16302,0
```

gives full screen graphics; that is, it extends the graphics area downward, over the area that could accommodate four lines of text in all the earlier programs. The maximum *y* value of a pixel in full screen graphics is 191.

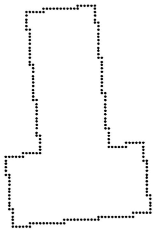


Figure 3.15

The statements

```
ROT=PDL(1)    and    SCALE=1+PDL(2)
```

set the rotation RO and scale SC from the "values" of the paddles 1 and 2 (that is, to numbers between 0 and 255 that depend on the rotation of the knobs). The statement

```
XDRAW 1 AT X,Y
```

erases shape 1 from its previous position before drawing it afresh. The hammer is displayed for the duration of the FOR loop that provides a short delay, and the cycle is repeated.

Suggested variations

1. Use the two paddles to control the *x* and *y* positions of a shape on the screen.
2. Control the *x* position, *y* position, scale, or rotation of the two shapes on the screen from the two paddles.
3. Use a paddle to control which shape, selected from a more extensive shape table, is on the screen.
4. Adapt ALOHA1 and ALOHA2 to use paddles to control the position of the building blocks.
5. Adapt DESIGN1 and DESIGN2 to use paddles to select raster points for inclusion in the building block that is being designed.
6. Extend the shape table and hammer program to let the user drive some imaginary nails into an imaginary board on the screen (Figure 3.16).

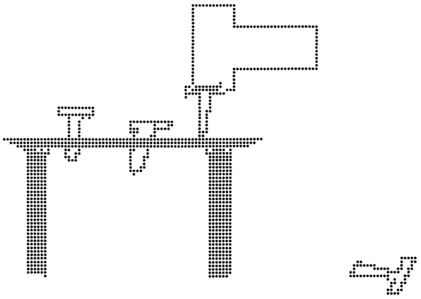


Figure 3.16

The HAMMER program

```
100  REM *** HAMMER-USES SHAPES AND PADDLE CONTROL ***
110  REM *** SET UP SHAPE TABLE ***
120  FOR LO=7676 TO 7690: READ PO: POKE LO,PO: NEXT LO
130  DATA 1,0,4,0,54,54,55,46,45,37,60,36,36,63,0
140  POKE 232,252: POKE 233,29: REM *** STORE BASE ADDRESS ***
150  HGR : POKE -16302,0: REM ***  FULL SCREEN GRAPHICS ***
160  RO=PDL(1):SC=PDL(0): IF SC=0 THEN SC=1
170  ROT=RO: SCALE=SC
180  XDRAW 1 AT 140,80: REM *** DRAW THE HAMMER ***
190  FOR DE=1 TO 30: NEXT
200  XDRAW 1 AT 140,80: REM *** ERASE THE HAMMER ***
210  GOTO 160
```

Chapter 4

Going around in circles

The two previous chapters described a great many things using points and straight lines. This chapter adds circles to the picture, and shows how these can be used to produce many more curves that are decorative, interesting, and useful. Readers who would like to find out or be reminded about polar coordinates will find a simple explanation, and the mathematically minded will find scope to experiment and innovate with a variety of geometrical constructions.

4.1 HOW TO DRAW CIRCLES

There is more than one way to skin a cat, and there are many ways to draw a circle, whether on paper, in the sand, or on a graphics screen. When a messenger came to the studio of a great Italian painter 500 years ago for a sample of his work to be used in awarding a commission, the artist just drew a large circle on a sheet of paper, which was so perfect that he got the contract. Preparing a bid usually takes a little more time—but perfection of circles on a graphics screen is a good sign of its quality—and the speed with which it is produced can often say a lot about the programmers.

Because the graphics screen consists of a raster of pixels, the best circle that you can possibly get will look a little jagged if it is inspected too closely. Figure 4.1 shows a "good" circle, rather magnified to emphasize this effect. The larger the radius of the circle in relation to the pixel size, the smoother it will appear.

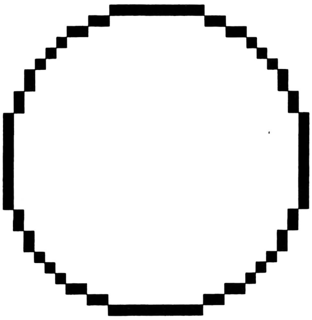


Figure 4.1

A regular polygon with a large number of sides is often used to approximate a circle in computer graphics. In Figure 4.2, see how these polygons look more and more like a circle as the number of sides increases. These polygons were drawn precisely. On inexpensive graphics screens some of the sides suffer from "jaggies" because the pixels are not infinitely small (see Figure 4.3).

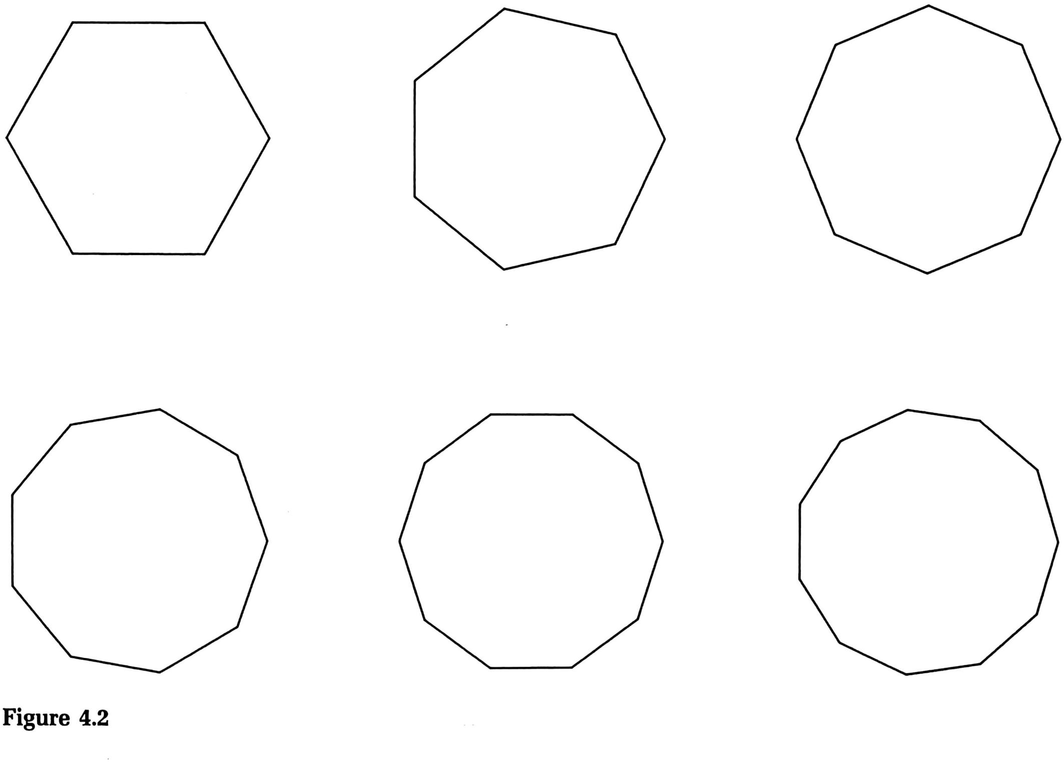


Figure 4.2

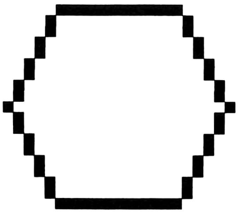


Figure 4.3

So much for background. Now we shall make the Apple draw some circles. The program CIRCLES prompts

```
TYPE THE RADIUS & COORDINATES OF THE CENTER:
```

and, when you have responded by typing the data (measured in screen coordinates), the program draws the circle that you requested. The x and y coordinates of the center cannot be less than the radius. The value of the x coordinate plus the radius cannot exceed the width of the screen (279), and the value of the y coordinate plus the radius cannot exceed the depth (159).

The main variables in the program are

`X0,Y0`	The coordinates of the center
`R`	The radius
`X,YU`	The coordinates of a point on the upper half of the circle
`X,YL`	The coordinates of a point on the lower half of the circle
`XP,PU`	The coordinates of the previous point on the upper half
`XP,PL`	The coordinates of the previous point on the lower half

The program is based on the first formula that you were taught for a circle,

$$(x-x_0)^2+(y-y_0)^2=r^2$$

This follows from the theorem of Pythagoras and the fact that all the points on the circle are equidistant from the center. It gives the quadratic equation

$$y^2-2y_0y+(x-x_0)^2+y_0{}^2-r^2=0$$

which, using the formulas to solve a quadratic equation, and to factorize a difference of squares, gives

$$y=y_0\pm\sqrt{(r-x+x_0)(r+x-x_0)}$$

This is used in statements 160–170 to compute the y values on the circumference from the successive values of x. These increase by 1 from X0−R to X0+R. The YU values on the upper half of the circle are found using the + sign before the square root. The YL values on the lower half are computed using the − sign.

This program is given first because it uses the mathematics of a circle, which are learned first. For many purposes the programs in the next two sections are superior for reasons that will be explained. CIRCLES is useful, however, to make a start, and also for comparison with later programs, as a reminder that a little more mathematics often can speed up programs enormously. Very large computers tended to obscure this. Getting the most from your personal computer, which you may have bought with your own money, makes the issue more visible again and more personal.

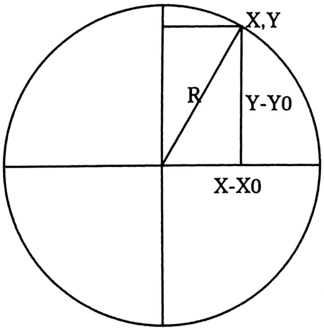


Figure 4.4

Suggested variations

1. Allow the user to specify the amount that x increases for the successive points that are plotted.
2. Draw a solid circle by drawing vertical lines between the points on the lower and upper halves of the circle with the same x value.
3. Allow the user to specify the inner and outer radii of a ring, and make the program draw the circles that form its inner and outer edges.
4. Make the program draw a solid ring.
5. Draw a circle that is shaded with vertical lines spaced a few units apart.
6. Draw a circle that is shaded with horizontal lines spaced a few units apart. (*Hint:* Make the FOR loop compute values of x on the left and right halves of the circle for successive values of y.)
7. Draw a square around the circle that just touches it.
8. Draw a square inside the circle that just touches it.
9. Allow several circles to be drawn on the screen and then erased.
10. Many computers add faster than they multiply. See if you can speed up the program by avoiding the multiplication to compute the number that is "square rooted" in statement 160. [*Hint:* This number is

    $$r^2-(x-x_0)^2=(r-x+x_0)\,(r+x-x_0).$$

    It was factorized to require just one multiplication. However, if

    $$s=r^2-(x-x_0)^2, \qquad \text{and} \qquad s_1=r^2-(x+1-x_0)^2,$$

    then

    $$s_1-s=2\,(x_0-x)-1$$

    This is the amount by which the argument of the square root increases in each cycle. Use this fact.]
11. Draw the part of a circle between values of x (or y) that are typed in response to a prompt.

The program CIRCLES

```
100  REM *** CIRCLES ***
110  ONERR  GOTO 210: REM *** USED TO BYPASS POINTS OF THE SCREEN ***
120  HGR : HCOLOR=3: HOME : VTAB 21
130  INPUT "TYPE THE RADIUS & COORDINATES OF THE CENTER: ";R,X0,Y0
```

```
140  IF X0-R>=0 AND X0+R<279 AND Y0>=0 AND Y0+R<=159 THEN 16
150  PRINT "PART OF THE CIRCLE WILL BE OFF THE SCREEN"
160  XP=X0-R:PU=Y0:PL=Y0:X=X0-R+1
170  RD=SQR((R-X+X0)*(R+X-X0))
180  YU=Y0+RD:YL=Y0-RD
190  IF YL<0 AND YL>-.01 THEN YL=0
200  HPLOT XP,PU TO X,YU: HPLOT XP,PL TO X,YL
210  XP=X:PU=YU:PL=YL
220  IF X<X0+R THEN X=X+1: GOTO 170
230  PRINT "PRESS ANY KEY TO CONTINUE";: GET A$: PRINT : GOTO 120
```

4.2 A SLIGHT CASE OF *JAWS*

The program MR. MOUTH displays a solid circle from which a wedge that constantly waxes and wanes is missing (Figure 4.5). With a little imagination and elaboration, this could be a mouth engaged in song or earnest conversation.

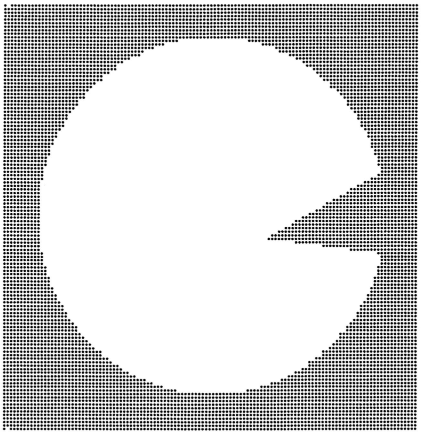


Figure 4.5

The program uses some elementary trigonometry—a subject that has been going for more than 2,000 years. An early Greek mathematician called Appolonius is credited with the first known work on it. If we draw spokes from the center to the points that are plotted evenly around a circular path, the angles between successive spokes are the same (Figure 4.6). It is convenient therefore to specify the position of a point on a circle by (1) the radius of the circle and (2) the angle that this radius makes with a horizontal line through the center, pointing toward the right. These are denoted traditionally by (1) the letter r (for radius) and (2) the Greek letter θ (theta), just as x and y are used for Cartesian coordinates.

When you rotate through a full circle you turn by 360°. If 10 points are spaced evenly around a circle, starting on a line through the center, they will have θ values of 0°, 36°, 72°, 108°, 144°, ..., 324°. These are just the successive multiples of 360/10°. The Applesoft functions actually use angles that are measured in radians, that is, multiples of $2\pi/10$ for the example in hand. To plot a point on the screen, its x and y values are needed. These are computed from the r and θ values by formulas that also contain the coordinates of the center, x_0 and y_0.

$$x=x_0+r\cos\theta, \qquad y=y_0+r\sin\theta$$

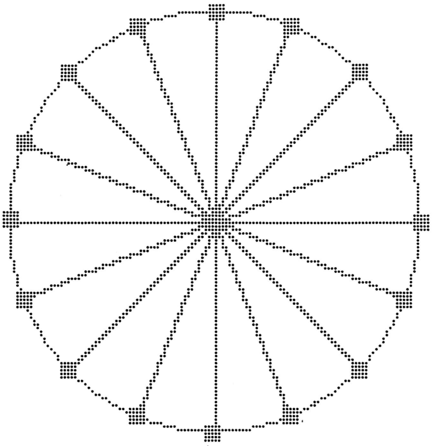


Figure 4.6

These formulas really repeat the definitions of sines and cosines that you learn at the very beginning of a trigonometry course (Figure 4.7). The sine of the base angle of a right-angled triangle is the ratio of the height to the hypotenuse, and the cosine of the angle is the ratio of the base to the hypotenuse.

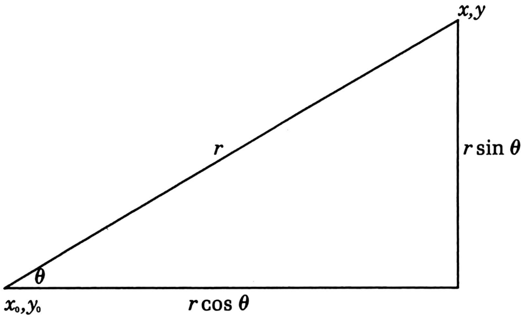


Figure 4.7

The main variables of the program are

`X0,Y0`	Coordinates of the center of the face
`XM,YM`	Coordinates of the joint between the jaws
`R`	Radius of the face
`TC`	Factor that limits the time the mouth stays closed
`TG`	Factor that limits the time the mouth stays open
`SO`	Factor that limits the maximum speed with which it opens
`MS`	Factor that limits the minimum speed with which it opens and closes
`TO`	Factor that limits the time it stays open
`SC`	Factor that limits the maximum speed with which it closes
`NO`	Factor that limits the angle to which the mouth opens
`X%(100)`	Array containing x coordinates of the circular outline
`Y%(100)`	Array containing y coordinates of the circular outline

N	Index of FOR loops that open and shut the mouth
K	Index of FOR loops that control the timing delays
TC%	Factor corresponding to TC for the current cycle
.	
.	
.	
NO%	Factor corresponding to NO for the current cycle

The program gets the constants X0, ..., NO from the DATA statement, constructs the coordinates of the outline, and draws the solid circle. It then begins the main cycle, in which it

1. Computes random values of the quantities TC%, SO%, ..., NO% from the limiting values TC, SO, ..., NO
2. Leaves the mouth closed for a time that depends on TC%
3. Opens it at a speed determined by SO%
4. Leaves it open for a time that depends on TO%
5. Closes it at a speed determined by SC%

The program then repeats this cycle, with new random values of the quantities TC% to NO%, and continues doing this until you type CONTROL C.

Suggested variations

1. Give the head some embellishments, such as a hat, an eye, a collar, or a cigar. (The mouth cannot completely close in the last case.)
2. Have two facing heads jabber in alternation or at the same time.
3. Make the program prompt the user for the constants that control the action.
4. Write a program to plot n points spaced uniformly around a circular path.
5. Draw n circular arcs, spaced evenly around a circular path, where the user specifies (1) n and (2) the relative lengths of the arcs that are lighted up and the arcs between them—for example, if n is 10 and the ratio is 2 to 1, the output is shown in Figure 4.8.

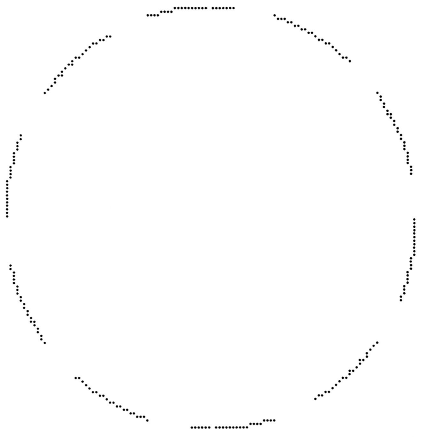


Figure 4.8

The MR. MOUTH program

```
100  REM *** MR. MOUTH ***
110  HOME : VTAB 12: PRINT "PLEASE WAIT ..."
120  DIM X%(314),Y%(314): REM *** ARRAYS FOR X AND Y VALUES ON A CIRCLE ***
130  READ X0,Y0,R,TC,SO,MS,TG,SC,NO: REM *** PARAMETERS DEFINING THE HEAD ***
140  DATA 140,80,50,10,10,10,10,10,35
150  XM=X0+R/3:YM=Y0+R/8: REM *** HINGE OF THE JAWS ***
160  TH=0:DT=.02
170  FOR N=0 TO 314: REM *** LOOP TO COMPUTE THE X AND Y VALUES ***
180  X%(N)=X0+R*COS(TH):Y%(N)=Y0+R*SIN(TH):TH=TH+DT
190  NEXT N
200  REM *** SWITCH TO HI-RES GRAPHICS AND TURN THE SCREEN TO WHITE ***
210  HGR : HCOLOR=3: HPLOT 1,1: CALL 62454: HCOLOR=0
220  REM *** DRAW THE SOLID CIRCLE ***
230  FOR N=0 TO 314
240  HPLOT X0,Y0 TO X%(N),Y%(N)
250  NEXT N
260  REM *** RESET THE RANDOM VARIABLES ***
270  TC%=INT(TC*RND(1)):SO%=INT(SO*RND(1))+MS
280  TG%=INT(TG*RND(1)):SC%=INT(SC*RND(1))+MS
290  NO%=INT(NO*RND(1))
300  REM *** KEEP THE MOUTH CLOSED ***
310  FOR K=1 TO 100*TC%: NEXT K
320  REM *** OPEN THE MOUTH ***
330  HCOLOR=3
340  FOR N=314 TO 314-NO% STEP -1
350  HPLOT XM,YM TO X%(314-N),Y%(314-N)
360  HPLOT XM,YM TO X%(N),Y%(N)
370  FOR K=1 TO 3*SO%: NEXT K
380  NEXT N
390  REM *** KEEP THE MOUTH OPEN ***
400  FOR K=100 TO 3*TG%: NEXT K
410  REM *** CLOSE THE MOUTH ***
420  HCOLOR=0
430  FOR N=314-NO% TO 314
440  HPLOT XM,YM TO X%(314-N),Y%(314-N)
450  HPLOT XM,YM TO X%(N),Y%(N)
460  FOR K=1 TO 3*SC%: NEXT K
470  NEXT N
480  REM *** REPEAT THE CYCLE ***
490  GOTO 260
```

4.3 JAZZING IT UP

Do you know any kids (or adults) with an Apple who want to immobilize their parents, teachers, friends, and relatives? SON OF MR. MOUTH is the ideal gift. It starts like the Pied Piper calling its followers from afar. Screams to turn the **!?!* thing off or dire threats involving banishment may be the first reaction, but then the rot starts to set in, and hours later the victims may still be transfixed, jibbering in front of the screen.

SON OF MR. MOUTH simply uses the PEEK command to make the speaker emit grunts, groans, and eldritch shrieks in synchronization (harmony is too strong a word) with the mouths on the screen.

Once you type RUN, you are at its mercy until CONTROL C is typed to stop it. Enough may be enough, or in this case too much. All the same, we cannot supress one more thought on the matter. If you have (i) sufficient gall, clout, or independence, (ii) symphonic or choral aspirations, and (iii) access to a room full of Apples...

The SON OF MR. MOUTH program

```
100  REM *** SON OF MR. MOUTH
110  DIM X%(314),Y%(314): REM *** ARRAYS FOR X AND Y VALUES ON A CIRCLE ***
120  H=-16336: REM ***PEEK ADDRESS TO MAKE A NOISE ***
130  READ X0,Y0,R,TC,SO,MS,TG,SC,NO: REM *** PARAMETERS DEFINING THE HEAD ***
140  DATA 140,80,50,10,10,10,10,10,35
150  TH=0:DT=.02
160  FOR N=0 TO 314: REM *** COMPUTE THE X AND Y VALUES ***
170  X%(N)=X0+R*COS(TH):Y%(N)=Y0+R*SIN(TH):TH=TH+DT
180  NEXT N
190  HOME : VTAB 21: HTAB 16: FLASH : PRINT "WELCOME!": NORMAL
200  REM *** CHANGE THE HI-RES SCREEN TO WHITE ***
210  HGR : HCOLOR=3
220  HPLOT 1,1: CALL 62454: HCOLOR=0
230  REM *** DRAW THE SOLID CIRCLE ***
240  FOR N=0 TO 314
250  HPLOT X0,Y0 TO X%(N),Y%(N)
260  NEXT N
270  REM *** PLOT THE EYE ***
280  HCOLOR=3: FOR MP=153 TO 158: HPLOT MP,55 TO MP,60: NEXT MP
290  HCOLOR=0
300  REM RESET THE RANDOM VARIABLES
310  TC%=INT(TC*RND(1)):SO%=INT(SO*RND(1))+MS
320  TG%=INT(TG*RND(1)):SC%=INT(SC*RND(1))+MS
330  NO%=INT(NO*RND(1))
340  REM *** KEEP THE MOUTH CLOSED ***
350  FOR K=1 TO 100*TC%: NEXT K
360  REM *** OPEN THE MOUTH ***
370  HCOLOR=3
380  FOR N=314 TO 314-NO% STEP -1
390  HPLOT X0,Y0 TO X%(314-N),Y%(314-N)
400  HPLOT X0,Y0 TO X%(N),Y%(N)
410  NN=INT(RND(1)*10)+1
420  FOR K=1 TO SO%: REM *** MAKE RANDOM SOUNDS ***
430  ON NN GOSUB 620,640,660,680,700,720,740,760,760,760
440  NEXT K
450  NEXT N
```

```
460  REM *** KEEP THE MOUTH OPEN ***
470  FOR K=100 TO 3*TG%: NEXT K
480  REM *** CLOSE THE MOUTH ***
490  HCOLOR=0
500  FOR N=314-N0% TO 314
510  HPLOT X0,Y0 TO X%(314-N),Y%(314-N)
520  HPLOT X0,Y0 TO X%(N),Y%(N)
530  NN=INT(RND(1)*10)+1
540  FOR K=1 TO SC%: REM *** MAKE RANDOM SOUNDS ***
550  ON NN GOSUB 620,640,660,680,700,720,740,760,770,770
560  NEXT K
570  NEXT N
580  REM *** REPEAT THE CYCLE ***
590  GOTO 300
600  END
610  REM *** SOUND SUBROUTINES ***
620  RR=PEEK(H)
630  RETURN
640  RR=PEEK(H)+PEEK(H)
650  RETURN
660  RR=PEEK(H)+PEEK(H)+PEEK(H)
670  RETURN
680  RR=PEEK(H)+PEEK(H)+PEEK(H)+PEEK(H)
690  RETURN
700  RR=PEEK(H):RR=PEEK(H):RR=PEEK(H)+PEEK(H)
710  RETURN
720  RR=PEEK(H)+PEEK(H):RR=PEEK(H)*PEEK(H)-PEEK(H)
730  RETURN
740  RR=PEEK(H)*PEEK(H)-PEEK(H)
750  RETURN
760  RR=PEEK(H)+14*1/14-PEEK(H):RR=PEEK(H)
770  RETURN
```

4.4 SQUARING AND DENTING A CIRCLE

You can get your own back on circles with the program CIRCLOIDS. It draws shapes such as those in Figures 4.9A–4.9E, which are determined by a number N. When N is 2, the shape is a circle (Figure 4.9A). When N is 1 the shape is a rhombus (that is, a square turned through 45°; Figure 4.9B). The closer N is to 0, the closer the curve approaches a pair of crossed lines. Conversely, the larger the value of N, the closer the curve comes to a square.

The program prompts for N and the "radius" of the curve. This actually is the distance from the center to the four most distant points—these are along the horizontal and vertical lines through the center, when N is not 2. It is the actual radius when N is 2. After you have responded, the program plots the curve and tells you to press a key to continue. It then repeats the initial prompt and the rest of the process that has been described, without erasing the screen.

The program simply plots the curve with the equation

$$|x-x_0|^n+|y-y_0|^n=r^n$$

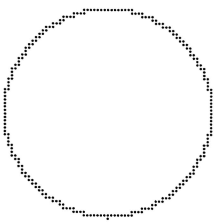


Figure 4.9A. N=2

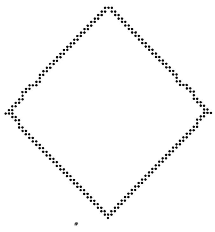


Figure 4.9B. N=1

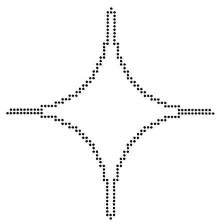


Figure 4.9C. N=0.25

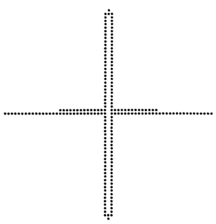


Figure 4.9D. N=0.1

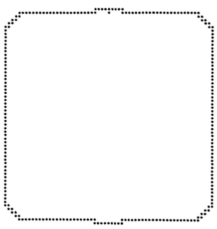


Figure 4.9E. N=10

When *n* is 2 this is the familiar equation for a circle, which we used in CIRCLES. The mathematically minded can show that the curve does approach the cross when *n* is very small and the square when *n* is very large.

The main variables in the program are

N	The value of *n* in the equation given above
R	The "radius" *r* in the equation given above
X0,Y0	The coordinates of the center, (x_0,y_0)
X,Y	The coordinates of the current point used for plotting in the first quadrant of the curve, relative to the center
XP,YP	The coordinates of the previous point used for plotting in the first quadrant of the curve, relative to the the center

For each value of X in the range 1–R the program computes the corresponding value of Y in the first quadrant of the curve. The line is drawn to this point from the point with the previous value of X. The corresponding lines are drawn in the other three quadrants by reversing the sign of X and/or Y.

Suggested variations

1. Allow the user to erase the screen or to keep the previous curves before plotting another.
2. Allow two concentric curves to be plotted and the area between them to be filled in.
3. Use the eightfold symmetry to compute coordinates from the equation for YC in just one eighth of the curve.
4. For the mathematically minded—construct the polar equation for the curve and use it.

The CIRCLOIDS program

```
100  REM *** CIRCLOIDS ***
110  X0=140:Y0=80
120  HGR : HCOLOR=3
130  HOME : VTAB 21
140  INPUT "RADIUS AND POWER? ";R,N
150  XP=0:YP=R
160  FOR X=1 TO R
170  Y=ABS(X ^ N-R ^ N) ^ (1/N)
180  HPLOT X0-X,Y0+Y TO X0-XP,Y0+YP
190  HPLOT X0-X,Y0-Y TO X0-XP,Y0-YP
```

```
200  HPLOT X0+X,Y0+Y TO X0+XP,Y0+YP
210  HPLOT X0+X,Y0-Y TO X0+XP,Y0-YP
220  XP=X:YP=Y
230  NEXT X
240  PRINT "PRESS ANY KEY TO CONTINUE";: GET A$: PRINT : GOTO 130
```

4.5 KIDNEYS, HEARTS AND SNAILS

Do you like kidneys, veal hearts, or snails? The program ENVELOPES produces such plots as those in Figures 4.10A–4.10C. Each of these plots is a "family" of circles with an "envelope," that is, an outline that is a curve with a special name and a special equation. The three envelopes in this figure are called, respectively, a cardioid, a nephroid, and a limaçon because they look like a heart, a kidney, and a snail (at least a seventeenth-century French snail, perhaps after the Three Musketeers trod on it.)

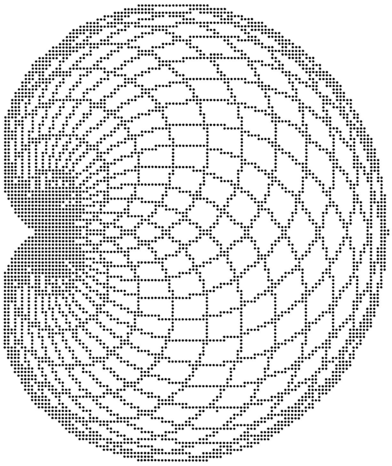


Figure 4.10A

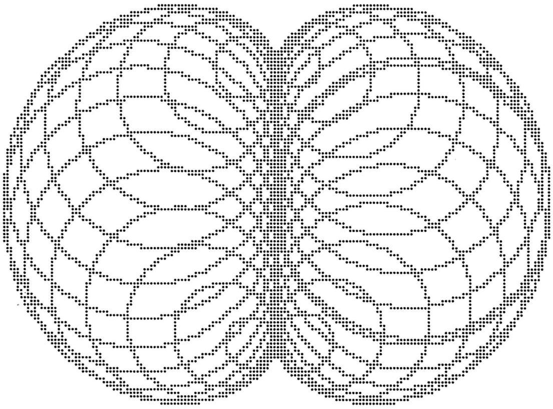


Figure 4.10B

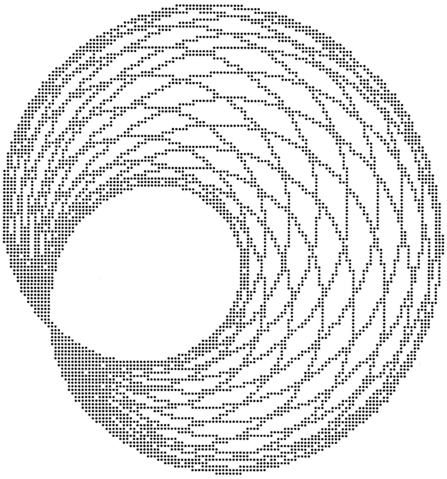


Figure 4.10C

The program begins the first plot by drawing a circle centered at (80,140) with radius 25. It then picks a point Q on this "base circle" and uses it as the center of another circle that it draws through the point P (Figure 4.11). The program then picks another point on the base circle, and uses this as the center of yet another circle that it draws through P (Figure 4.12). The program draws a succession of further circles, all of which have their centers on the base circle, and all of which go through P. The points that it uses as centers are spaced uniformly around the base circle, and the first plot of this section results. The point P is called the cusp point.

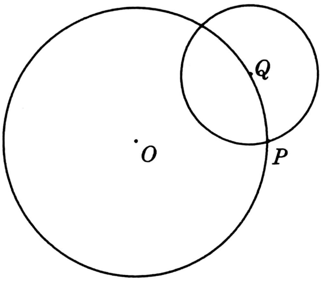


Figure 4.11

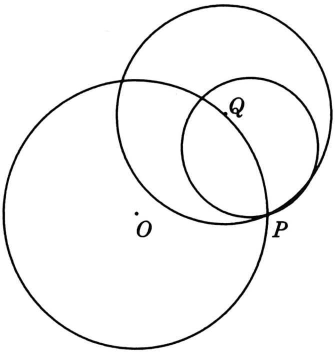


Figure 4.12

The second family of circles, which block out a nephroid, uses a base circle and its vertical diameter (Figure 4.13). The program picks a point on the base circle, and uses it as the center of another circle that it draws, which touches the vertical diameter (Figure 4.14). The program picks another point on the base circle, and uses it as the center of yet another circle that it draws, which also touches the original vertical diameter (Figure 4.15).

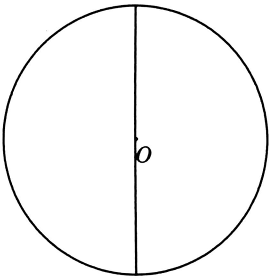


Figure 4.13

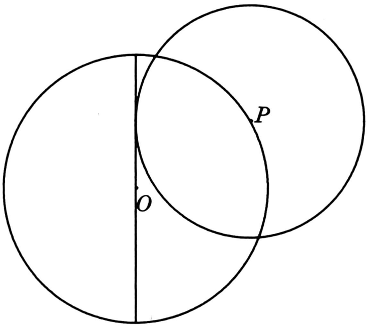


Figure 4.14

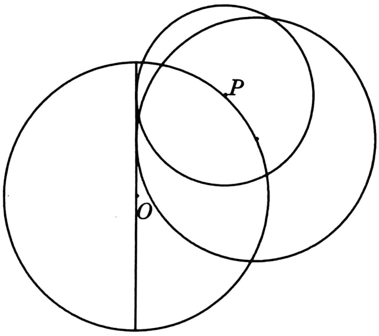


Figure 4.15

The third of the plots, which has a limaçon as its outline, uses a base circle and the point P on the horizontal line through its center (Figure 4.16). Each of the circles that block out the limaçon has its center on the base circle and goes through P (Figure 4.17). If P is on the base circle, the limaçon simply becomes a cardioid.

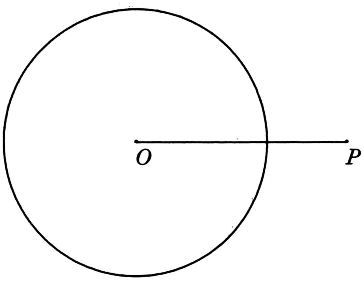


Figure 4.16

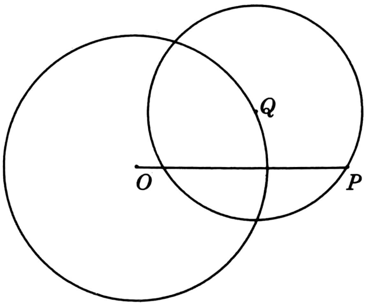


Figure 4.17

The formulas for drawing these families of circles and lines are very simple. For the cardioid, nephroid, and limaçon, draw axes through the center of the base circle. The angle θ takes successive values from 0° to 360° for each family of circles. Let the radius of the base circle be R (see Figures 4.18A–4.18C). The center of the circle that is drawn in the cycle for successive values of θ has the coordinates

$$x = R\cos\theta, \qquad y = R\sin\theta$$

relative to the center of the base circle. The radius of the circle that is drawn in the θ cycle is given by

$r = 2R\cos(\theta/2)$ for the cardioid

$r = R\cos\theta$ for the nephroid

$r = \sqrt{(R^2 + a^2 - 2aR\cos\theta)}$ for the limaçon

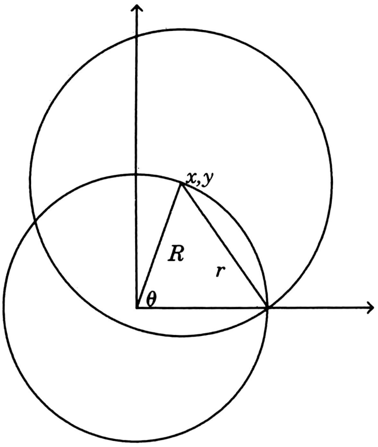


Figure 4.18A

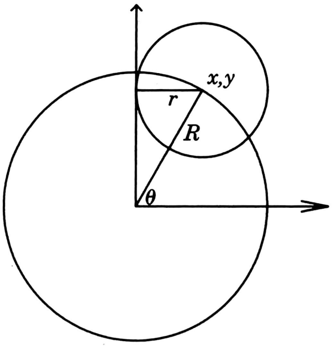


Figure 4.18B

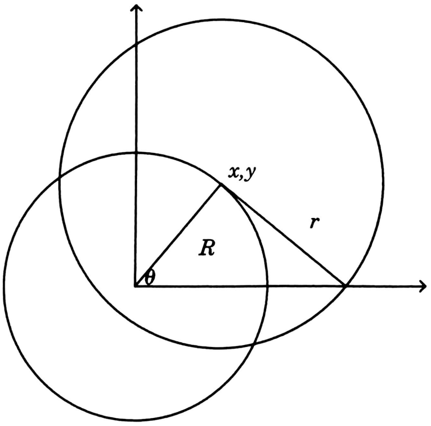


Figure 4.18C

The ENVELOPES program also lets you draw a family of straight lines that has an envelope called an astroid (Figure 4.19). All the lines are the same length, and go from the x axis to the y axis. Each line in the first quadrant just joins a point $(x,0)$ on the x axis to $(0,y)$ on the y axis. If L is the length of these lines,

$$y = \sqrt{L^2 - x^2}$$

by Pythagoras's theorem (Figure 4.20). These lines are drawn for x taking values from 0 to 1 in uniform steps. In the second, third, and fourth quadrants, the lines join $(-x,0)$ to $(0,y)$, $(-x,0)$ to $(0,-y)$, and $(x,0)$ to $(0,-y)$.

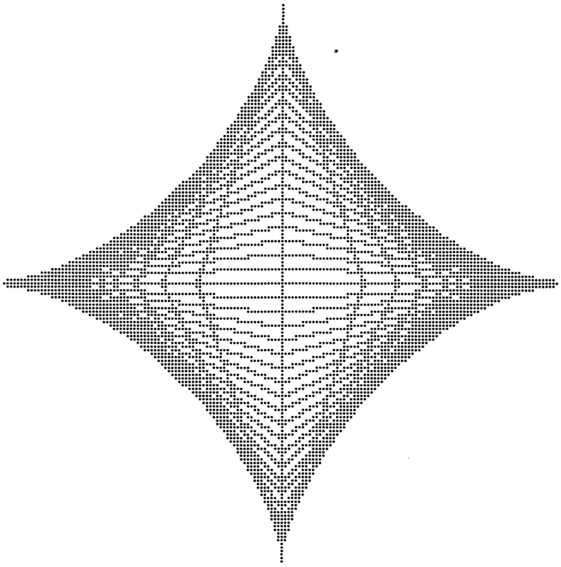


Figure 4.19

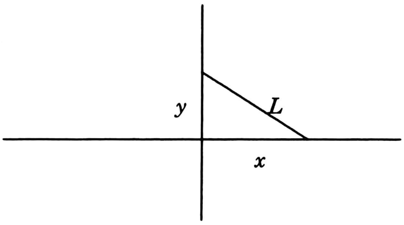


Figure 4.20

When the program prompts for data, it lets you clear the screen or draw the next family of curves on top of the previous one. Putting an astroid on top of a nephroid gives an interesting pattern (Figure 4.21).

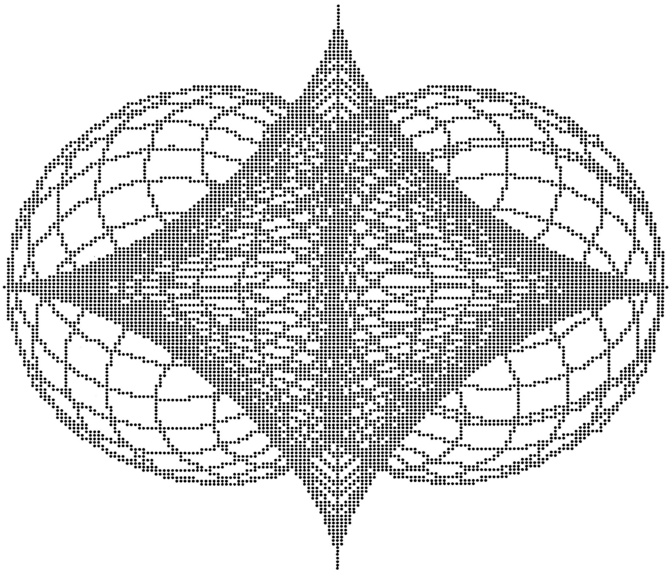


Figure 4.21

The ENVELOPES program begins in the same spirit as MR. MOUTH, by computing the x and y coordinates for fixed points spaced evenly around an eighth of a prototype circle. It then displays the options (1–4 to block out a nephroid, cardioid, limaçon, or astroid, respectively, and 5 to quit), in "inverse" mode (that is, reverse video), and tells the user to type whichever is needed. It asks whether the picture should be superimposed on the previous one. When a limaçon has been requested, the program prompts for the coordinates of the cusp in screen coordinates. For the families that produce a cardioid, nephroid, or limaçon, it prompts for the step size used to plot the individual circles (1 gives 320 points, 2 gives 160 points, 3 gives 108 points, and so forth). It goes to the circle-plotting subroutine repeatedly, from within the appropriate FOR loop, to construct the required output. It then returns to the main prompt. For an astroid, it draws the required lines from a succession of x values on the x axis to the appropriate y values on the y axis.

The main variables in the program are

`X(40)`	The x coordinates of points in an octant of the prototype circle
`NA`	Indexes the X and Y arrays
`Y(40)`	The corresponding y coordinates
`OP`	The option number
`ST`	Step size to plot the circles—the program sets this to 8 if the user types a larger value
`TH`	Angle, around the base circle, of the center of a circle that is part of the family being drawn
`XM,YM`	The x, y coordinates of the center of this circle
`RM`	Radius of this circle
`XC,YC`	Coordinates of the cusp of a limaçon and of the center of an astroid
`XN,YN`	Coordinates of the current point to be plotted on a circle, measured from its center
`XP,YP`	Corresponding coordinates of the previous point to be plotted

The overall structure of the program is straightforward. The portions of the program for the different kinds of family simply correspond to the formulas that have been explained.

Suggested variations

1. Allow the user to specify the radius and center of the families of curves.
2. Allow the position of the cusp to be varied in the cardioid and limaçon families.
3. Allow the line used to define a nephroid family to be at an angle specified by the user.
4. Allow the axes of an astroid family to be at an angle specified by the user.

The ENVELOPES program

```
100  REM *** ENVELOPES-PRODUCES CARDIOIDS ETC ***
110  TEXT : HOME : VTAB 12: PRINT "<PLEASE WAIT-COMPUTING>"
120  NA=1: DIM X(40),Y(40): REM *** FOR POINTS ON A PROTOTYPE CIRCLE ***
130  FOR A=0 TO .8 STEP .02:X(NA)=COS(A):Y(NA)=SIN(A):NA=NA+1: NEXT A
140  TEXT : HOME : INVERSE
150  VTAB 4:REM *** DISPLAY MAIN MENU
160  HTAB 16: PRINT "OPTIONS": PRINT : PRINT : PRINT
170  FOR LO=1 TO 5: HTAB 6: READ A$: PRINT A$: NEXT : RESTORE
180  NORMAL
190  DATA  1. NEPHROID,2. CARDIOID,3. LIMACON,4. ASTROID,5. QUIT
200  PRINT : PRINT : HTAB 10: INPUT "WHICH OPTION? ";OP
210  HOME : VTAB 21
230  ON OP GOTO 300,400,500,600,680
240  GOTO 130
300  REM *** NEPHROID ***
310  GOSUB 1160
330  INPUT "ENTER THE STEP SIZE? ";ST: IF ST>8 THEN ST=8
340  FOR TH=0 TO 6.28 STEP .195
350  XM=140+40*COS(TH):YM=80+40*SIN(TH):RM=140-XM: GOSUB 1010
360  NEXT TH
370  GOTO 1200
400  REM *** CARDIOID ***
410  GOSUB 1160
430  INPUT "ENTER THE STEP SIZE? ";ST: IF ST>8 THEN ST=8
440  FOR TH=0 TO 6.28 STEP 6.28/27:XM=100+25*COS(TH)
450  YM=80+25*SIN(TH):RM=((XM-75) ^ 2+(YM-80) ^ 2) ^ .5
460  GOSUB 1010
470  NEXT TH
480  GOTO 1200
500  REM *** LIMACON ***
510  GOSUB 1160
530  INPUT "ENTER THE STEP SIZE? ";ST: IF ST>8 THEN ST=8
540  INPUT "ENTER THE CUSP COORDINATES? ";XC,YC
550  FOR TH=0 TO 6.28 STEP .32:XM=140+15*COS(TH)
560  YM=80+15*SIN(TH):RM=((XM-XC) ^ 2+(YM-YC) ^ 2) ^ .5
570  GOSUB 1010
580  NEXT TH
590  GOTO 1200
600  REM *** ASTROID ***
610  GOSUB 1160
630  INPUT "ENTER LENGTH AND STEP? ";L,ST
640  INPUT "ENTER THE CENTER COORDINATES? ";XC,YC
650  FOR YP=YC-L TO YC+L STEP ST:X=(L ^ 2-(YP-YC) ^ 2) ^ .5
660  HPLOT XC,YP TO XC+X,YC: HPLOT XC,YP TO XC-X,YC: NEXT YP
670  GOTO 1200
680  END
```

```
1000  REM *** DRAW A CIRCLE USING 8-FOLD SYMMETRY ***
1010  XP=RM*X(1):YP=RM*Y(1)
1020  NA=2:SW=0
1030  XN=RM*X(NA):YN=RM*Y(NA)
1040  HPLOT XM+XN,YM+YN TO XM+XP,YM+YP
1050  HPLOT XM-XN,YM+YN TO XM-XP,YM+YP
1060  HPLOT XM+XN,YM-YN TO XM+XP,YM-YP
1070  HPLOT XM-XN,YM-YN TO XM-XP,YM-YP
1080  HPLOT XM-YN,YM-XN TO XM-YP,YM-XP
1090  HPLOT XM+YN,YM-XN TO XM+YP,YM-XP
1100  HPLOT XM-YN,YM+XN TO XM-YP,YM+XP
1110  HPLOT XM+YN,YM+XN TO XM+YP,YM+XP
1120  XP=XN:YP=YN
1125  IF SW THEN 1140
1130  NA=NA+ST: IF NA>=40 THEN NA=40:SW=1
1135  GOTO 1030
1140  RETURN
1150  REM *** PROMPT FOR INPUT COMMON TO ALL OPTIONS ***
1160  POKE -16304,0: POKE -16297,0
1170  INPUT "SUPERIMPOSE ON PREVIOUS BACKGROUND? ";SP$
1180  IF LEFT$(SP$,1)="N" THEN  HGR
1190  RETURN
1200  PRINT "HIT ANY KEY TO CONTINUE";: GET A$: PRINT : GOTO 140
```

Chapter 5

A little plotting and scheming

5.1 GRAPHICS FOR MANAGEMENT

Helping you to convince management of progress, needs, and the fact that you have firm control of what you are doing are among the most important roles of personal graphics in corporate life. Timely, accurate, and stylish slides and transparencies can be made for decision-making presentations to top executives of your own organization and to customers, clients, and staff. Graphs and charts can be prepared for company reports, local or world wide distribution, and bids and proposals. Business graphics can be transmitted electronically, adding a new dimension to document transfer and teleconferencing. Graphs and charts will be used increasingly, too, to give instant summaries and correlations of sales, markets, and other business data in response to on-line questions.

Graphs, bar charts, histograms, and pie charts are major components of business graphics presentations. Designing these is an art in itself—the subject of many articles and books and many software packages. Seeing some simple programs that draw graphs and charts can help you realize what you are paying for when you buy some of these packages. And there are applications that need simple plots combined with other elements that are easier to program from prototypes than to try fitting to a package that you purchased (or vice versa.)

Captions and labels are of paramount importance in business graphics. You can control the appearance and placement of text on a graphic display by making the computer draw each character precisely as it is wanted, within the resolution of the screen. The high resolution graphics generator of the DOS Tool Kit provides this capability. For readers who do not have the Tool Kit or who want to explore on their own, the program FONT FORMER in the next section introduces the technique that is used in the programs to plot graphs, bar charts, and pie charts, which are described in the following three sections of this chapter.

5.2 CREATING A FONT FILE

The program FONT FORMER lets you create text files of data that define different fonts needed to put words and symbols on a graphics display. To define a set of characters, you first draw them on graph paper, for example, on 5×7 rasters. You then find the ends of the lines that can be used in HPLOT instructions to display the characters. For example, an A can be drawn with its lower left corner at position (100,100) on the screen by

```
HPLOT 100,100 TO 100,96 TO 102,94 TO 104,96 TO 104,100
HPLOT 100,98 TO 104,98
```

Figure 5.1

Data that can be used in the graphics mode to draw the font that usually appears in the text mode of the Apple are listed at the end of this section. Each character is drawn on a 5×7 raster. The A is coded as two "segments," which correspond to the HPLOT statements displayed above. The first segment consists of five points that match the points in the first HPLOT statement. The second segment consists of two points that match the points in the second statement. The points are defined by their coordinates in the raster, with x taking values 0 to 4 (from left to right), and y taking values 0 to –6 (from bottom to top). The full coding for the A can be written as follows:

- Number of segments: 2
- Segment 1:
  Number of points: 5
  Coordinates: 0,0; 0,–4; 2,–6; 4,–4; 4,0
- Segment 2:
  Number of points: 2
  Coordinates: 0, –2; 4,–2

The numbers are strung out, with the number of segments first and then, for each segment, the number of points and the actual coordinates. For A, the list of numbers is

2, 5, 0, 0, 0, −4, 2, −6, 4, −4, 4, 0, 2, 0, −2, 4, −2

You can adapt this approach to use shape tables, but that takes a little more work (without an automatic shape table encoder), and may require far more points for letters that are drawn on a larger scale.

When the characters that you want to use have been defined, you can type the lists of numbers for them into the FONT FORMER program. This provides a check on the coding, and helps you with it. The program begins by displaying the menu:

```
1. DEFINE CHARACTERS
2. WRITE A TEXT FILE
3. READ A TEXT FILE
4. DISPLAY A STRING
5. QUIT
   PLEASE TYPE OPTION:
```

You respond by typing 1. The menu disappears, and the prompt

```
NUMBER OF CHARACTERS:
```

appears at the bottom of the screen. Type the number of characters you want to define. A new prompt appears:

```
CHARACTER NO. 1
NUMBER OF SEGMENTS:
```

If you are just defining an alphabet, type 2 (for the number of segments in the A). The prompt changes to

```
CHARACTER NO. 1
SEGMENT NO. 1
NUMBER OF POINTS:
```

Type 5 for the first segment of the A. The program prompts for the coordinates of the five points in turn. It then displays the segment at the center of the screen, and prompts for the number of points in the next segment. When this is complete, it is displayed as well, and the entire A remains on the screen until you press RETURN (or any other key.) The A disappears, and the program prompts for the number of segments in the second character. The cycle is repeated for the number of characters you said you would define. The program then displays the main menu again.

You can now type 2, to save the codes for the entire set of characters as a text file. The program prompts for the name you want to give it. The file is written, and the main menu reappears. This file can be used by the programs in the next three sections, and many other programs you may want to write. To check that it is correct, type 3. The program prompts for the name of the file. When you have typed this, the screen clears, the file is read, and the characters are displayed across the screen. Pressing any key brings the main menu back.

To display a string in the font that has been defined, type 4. The program prompts for (1) the screen coordinates of the point where you want the string to start, (2) the direction in which you want it written (that is, one of the numbers 1–4 for left to right, right to left, downward, and upward, respectively), (3) the direction you want the characters drawn (that is, one of the numbers 1–4 for the character rotated by 0°, 90°, 180°, and 270°, respectively, or one of the numbers

5–8 for each character to be reflected about its vertical median and rotated correspondingly), and (4) the actual string. For the string ABCDEFG, the numbers 1–8, typed in response to the prompt (3), display it as shown in Figure 5.2. To be consistent with the way FONT FORMER and the next three programs use the coded data, the position of a character in the font should be 31 less than the ASCII code of the key that you want to press to display it. You can skip characters that you do not want to define by simply typing 0 for the number of segments.The & and @ symbols were omitted in this way.

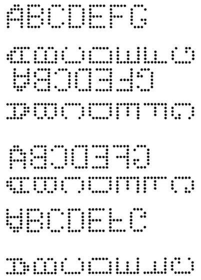


Figure 5.2

The data at the end of the section produced the character set shown in Figure 5.3. It corresponds to the ASCII codes as follows:

character	sp	!	"	#	$	%	&	'	(	)	*	+	,	–	.	/	0
ASCII	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
character	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?	@	A
ASCII	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65
character	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
ASCII	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82
character	S	T	U	V	W	X	Y	Z	[	\	]	^	_				
ASCII	83	84	85	86	87	88	89	90	91	92	93	94	95				

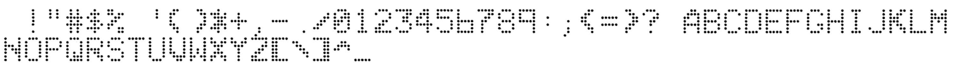


Figure 5.3

The main variables in the program are as follows

NC	The number of characters in the font
CN	Takes values 1, 2, 3, ... for successive characters in the font
NS	The number of segments in a character
SN	Takes values 1, 2, 3, ... for successive segments of the characters in the font, increasing from character to character, that is, not restarting at 1 for each
NP	The number of points in a segment

`PN`	Takes values 1, 2, 3, ... for successive points that define the characters in the font, increasing from segment to segment, that is, not restarting at 1 for each
`PT%(PN,1)`	The x coordinate of the PNth point in the raster that is used to define a character
`PT%(PN,2)`	The y coordinate of the PNth point in the raster
`DS%(SN)`	The PN value of the last point in the SNth segment
`DC%(CN)`	The SN value of the last segment of the CNth character
`XL`	The x coordinate of a letter on the screen
`YL`	The y coordinate of a letter on the screen
`M1`	Code for the direction in which a string is written
`M2`	Code for the orientation in which a character is drawn
`OP`	The option selected from the main menu
`S1`	The SN value of the first segment of the character that is being defined, written, or read
`S2`	The SN value of its last segment
`P1`	The PN value of the first point of the segment that is being defined, written, or read
`P2`	The PN value of its final point
`A4$`	The CONTROL D character
`FL$`	The name of the text file that is being written or read
`I`	The position of a character in a string that is being displayed
`H%`	The height of the character raster in the normal lettering mode
`W%`	The width of the character raster in this mode
`H,W`	The height and width of the raster on which the characters are actually drawn

One array, DC%, thus serves as the directory to a second array, DS%; it in turn serves as the directory to the array PT%, which contains the detailed plotting data. The program thus provides another illustration of the use of this kind of data structure. Also used in the program DESIGN2 (see Section 3.4), such structures have widespread application in graphics and other work.

The operation of the program is best understood by first following it when (1) a font is being defined (OP is 1), (2) a text file is being written (OP is 2), (3) a text file is being read (OP is 3), and (4) a string is being displayed (OP is 4). The processes in each are straightforward.

The statements that prompt for a font definition include several details concerned with screen formatting. The statement

```
POKE 34,23
```

restricts scrolling to the last two lines of the text area. This preserves the character and segment numbers in the first two lines, when data for the points are typed. The statement

```
CALL -958
```

erases the portion of the screen that follows the cursor. If it were omitted, part of the old prompts would remain on the screen when new prompts were given. The variations in the HPLOT operands for different orientations of characters follow from simple geometrical considerations.

Suggested variations

1. If you do not have a disk drive, and/or need only a small character set, include the codes in DATA statements, and use READ statements to transfer them to arrays or to access appropriate portions.
2. Allow the user to cancel a character that is being designed, and to restart it, without affecting the earlier characters in the font. [*Hint:* After line 540 prompt for a code to keep or cancel. Reset P1 to DS%(S1-1) to cancel.]
3. Allow the user to specify the raster size. [*Hint:* After line 420 set a prompt for W% and H%, set H and YL to H%, and set W to W%. Insert PRINT statements in line 380 to write W% and H% to disk.]
4. Design some other fonts, in Roman and non-Roman alphabets, and store them.
5. Extend the principle to use mixed fonts (for example, different sizes and subscripts and superscripts).
6. Combine the principle with the building block tactics of Chapter 3, using different keys to represent different building blocks.
7. Apply the principle to chemical and mathematical formulas.
8. Allow characters to be selected from different text files to form new ones.
9. Adapt the program to use shape definitions.

The FONT FORMER program

```
100  REM *** FONT FORMER ***
110  LOMEM: 16384: REM *** PROTECT MEMORY ***
120  DIM DC%(70),DS%(200),PT%(360,2)
130  W%=6:H%=8:A4$=CHR$(4)
140  TEXT : HOME :H=H%:W=W%: REM *** MAIN MENU ***
150  FOR I=1 TO 5: READ CA$: HTAB 10: PRINT I;".  ";CA$: NEXT I
160  DATA DEFINE CHARACTERS,WRITE A TEXT FILE,READ A TEXT FILE
170  DATA DISPLAY A STRING,QUIT
180  RESTORE
190  PRINT : INPUT "   PLEASE TYPE OPTION: ";OP
200  ON OP GOTO 230,350,300,590,790
210  PRINT "INVALID OPTION": GOTO 190
220  REM ***SET THE LETTERING MODE ***
230  XL=150:YL=70:M2=1
240  REM *** INITIALIZE SOME PROMPTS ***
250  B1$="NUMBER OF CHARACTERS: "
260  B2$="NUMBER OF SEGMENTS: "
270  B3$="NUMBER OF POINTS: "
280  B4$="COORDINATES: "
290  GOTO 410
```

```
300 REM *** SETS THE LETTERING MODE ***
310 XL=0:YL=H:M1=1:M2=1
320 REM *** NULLIFY SOME PROMPTS ***
330 B1$="":B2$="":B3$="":B4$=""
340 REM *** OPEN A FILE ***
350 HOME : INPUT "NAME OF FILE: ";FL$
360 PRINT A4$;"OPEN ";FL$
370 IF OP=3 THEN 390
380 PRINT A4$;"WRITE ";FL$: PRINT NC: GOTO 440
390 PRINT A4$;"READ ";FL$: HGR : HCOLOR=3: GOTO 420
400 REM *** GET THE NO. OF CHARACTERS ***
410 HOME : VTAB 21
420 PRINT B1$;: INPUT "";NC
430 REM *** CYCLE THROUGH THE CHARACTERS ***
440 S2=0:P2=0
450 FOR CN=1 TO NC
460 S1=S2+1
470 ON OP GOTO 480,520,500
480 HGR : HOME : VTAB 21
490 PRINT "CHARACTER NO. ";CN
500 PRINT B2$;: INPUT "";NS
510 S2=S1+NS-1:DC%(CN)=S2: GOTO 530
520 S2=DC%(CN):NS=S2-S1+1: PRINT NS
530 GOSUB 810
540 NEXT CN: REM *** END OF CYCLE ***
550 IF OP=1 THEN 140
560 PRINT A4$;"CLOSE ";FL$
570 GET ZZ$
580 GOTO 140
590 HGR : HCOLOR=3
600 VTAB 21: CALL -958: INPUT "POSITION OF STRING: ";XL,YL
610 INPUT "DIRECTION OF WRITING: ";M1
620 INPUT "DIRECTION OF LETTER: ";M2
630 POKE 34,23: INPUT "STRING: ";ST$
640 PRINT "STRING: ";ST$;: POKE 34,20
650 IF INT(M2/2)<>M2/2 THEN W=W%:H=H%
660 IF INT(M2/2)=M2/2 THEN H=W%:W=H%
670 IF LEN(ST$)<1 THEN 740
680 FOR I=1 TO LEN(ST$)
690 CN=ASC(MID$(ST$,I,1))-31
700 S1=DC%(CN-1)+1:S2=DC%(CN)
710 P2=DS%(S1-1)
720 NS=S2-S1+1: GOSUB 810
730 NEXT I
740 HOME
750 PRINT "TYPE 1 TO DISPLAY ANOTHER STRING, "
760 INPUT "     2 TO RETURN TO MAIN MENU: ";OQ
770 ON OQ GOTO 600,140
```

```
 780  GOTO 140
 790  END : REM *** QUIT ***
 800  REM *** DISPLAY A CHARACTER ***
 810  IF NS<1 THEN 1500
 820  REM *** CYCLE THROUGH THE SEGMENTS ***
 830  FOR SN=S1 TO S2
 840  P1=P2+1
 850  ON OP GOTO 860,930,890,930
 860  HOME : VTAB 21: PRINT "CHARACTER NO. ";CN
 870  PRINT "SEGMENT NO. ";SN-S1+1
 880  VTAB 23: CALL -958
 890  PRINT B3$;: INPUT "";NP
 900  P2=P1+NP-1
 910  DS%(SN)=P2
 920  GOTO 970
 930  P2=DS%(SN):NP=P2-P1+1: IF OP=2 THEN  PRINT NP
 940  GOTO 970
 950  P2=DS%(SN): GOTO 970
 960  REM *** CYCLE THROUGH THE POINTS ***
 970  FOR PN=P1 TO P2
 980  ON OP GOTO 990,1460,1010,1030
 990  PRINT B4$;: INPUT "";PT%(PN,1),PT%(PN,2)
1000  GOTO 1020
1010  INPUT "";PT%(PN,1): INPUT "";PT%(PN,2)
1020  REM *** DISPLAY A SEGMENT ***
1030  ON M2 GOTO 1050,1100,1150,1200,1250,1300,1350,1400
1040  STOP : REM *** INVALID LETTERING MODE ***
1050  IF NP=1 THEN  HPLOT XL+PT%(PN,1),YL+PT%(PN,2)
1060  IF PN<=P1 THEN 1470
1070  OX=PT%(PN-1,1)+XL:OY=YL+PT%(PN-1,2)
1080  HPLOT OX,OY TO PT%(PN,1)+XL,PT%(PN,2)+YL
1090  GOTO 1470
1100  IF NP=1 THEN  HPLOT XL+H+PT%(PN,2),YL-PT%(PN,1)
1110  IF PN<=P1 THEN 1470
1120  OX=XL+H+PT%(PN-1,2):OY=YL-PT%(PN-1,1)
1130  HPLOT OX,OY TO XL+H+PT%(PN,2),YL-PT%(PN,1)
1140  GOTO 1470
1150  IF NP=1 THEN  HPLOT XL+W-PT%(PN,1),YL-H-PT%(PN,2)
1160  IF PN<=P1 THEN 1470
1170  OX=XL+W-PT%(PN-1,1):OY=YL-H-PT%(PN-1,2)
1180  HPLOT OX,OY TO XL+W-PT%(PN,1),YL-H-PT%(PN,2)
1190  GOTO 1470
1200  IF NP=1 THEN  HPLOT XL-PT%(PN,2),YL-W+PT%(PN,1)
1210  IF PN<=P1 THEN 1470
1220  OX=XL-PT%(PN-1,2):OY=YL-W+PT%(PN-1,1)
1230  HPLOT OX,OY TO XL-PT%(PN,2),YL-W+PT%(PN,1)
1240  GOTO 1470
1250  IF NP=1 THEN  HPLOT XL+W-PT%(PN,1)-1,YL+PT%(PN,2)
1260  IF PN<=P1 THEN 1470
```

```
1270  OX=XL+W-PT%(PN-1,1)-1:OY=YL+PT%(PN-1,2)
1280  HPLOT OX,OY TO XL+W-PT%(PN,1)-1,YL+PT%(PN,2)
1290  GOTO 1470
1300  IF NP=1 THEN  HPLOT XL+H+PT%(PN,2),YL-W+PT%(PN,1)+1
1310  IF PN<=P1 THEN 1470
1320  OX=XL+H+PT%(PN-1,2):OY=YL-W+PT%(PN-1,1)+1
1330  HPLOT OX,OY TO XL+H+PT%(PN,2),YL-W+PT%(PN,1)+1
1340  GOTO 1470
1350  IF NP=1 THEN  HPLOT XL+PT%(PN,1),YL-H-PT%(PN,2)
1360  IF PN<=P1 THEN 1470
1370  OX=XL+PT%(PN-1,1):OY=YL-H-PT%(PN-1,2)
1380  HPLOT OX,OY TO XL+PT%(PN,1),YL-H-PT%(PN,2)
1390  GOTO 1470
1400  IF NP=1 THEN  HPLOT XL-PT%(PN,2),YL-PT%(PN,1)
1410  IF PN<=P1 THEN 1470
1420  OX=XL-PT%(PN-1,2):OY=YL-PT%(PN-1,1)
1430  HPLOT OX,OY TO XL-PT%(PN,2),YL-PT%(PN,1)
1440  GOTO 1470
1450  REM *** WRITE COORDINATES TO DISK ***
1460  PRINT PT%(PN,1): PRINT PT%(PN,2)
1470  NEXT PN,SN
1480  ON OP GOTO 1490,1670,1500,1500
1490  GET ZZ$: GOTO 1670
1500  REM *** PROVIDE THE ESCAPEMENT ***
1510  ON M1 GOTO 1550,1580,1610,1640
1520  PRINT "INVALID LETTERING DIRECTION"
1530  PRINT "PRESS ANY KEY AND TRY AGAIN"
1540  GET A$: GOTO 600
1550  IF XL>274-W THEN 1570
1560  XL=XL+W: GOTO 1670
1570  XL=0:YL=YL+H: GOTO 1670
1580  IF XL<W THEN 1600
1590  XL=XL-W: GOTO 1670
1600  XL=280-W:YL=YL+H: GOTO 1670
1610  IF YL>160-H THEN 1630
1620  YL=YL+H: GOTO 1670
1630  YL=0:XL=XL+W: GOTO 1670
1640  IF YL<H THEN 1660
1650  YL=YL-H: GOTO 1670
1660  YL=160-H:XL=XL+W: GOTO 1670
1670  RETURN
1680  REM *** THE END ***
```

Codes for a simple font

```
sp 0
!   2,1,2,0,2,2,-2,2,-6
"   2,2,1,-6,1,-4,2,3,-6,3,-4
#   4,2,1,0,1,-6,2,3,0,3,-6,2,0,-2,4,-2,2,0,-4,4,-4
```

$	6,2,0,-1,3,-1,1,4,-2,2,1,-3,3,-3,1,0,-4,2,1,-5,3,-5,2,2,-6,2,0
%	5,2,4,-5,0,-1,2,0,-6,1,-6,2,0,-5,1,-5,2,4,0,3,0,2,4,-1,3,-1
&	0
'	1,2,2,-6,2,-4
(	1,4,2,-6,0,-4,0,-2,2,0
)	1,4,2,-6,4,-4,4,-2,2,0
*	3,2,2,0,2,-6,2,0,-5,4,-1,2,0,-1,4,-5
+	2,2,0,-3,4,-3,2,2,-1,2,-5
,	2,1,1,1,2,2,0,2,-1
-	1,2,0,-3,4,-3
.	1,1,3,0
/	1,2,0,-1,4,-5
0	5,2,0,-1,0,-5,2,3,-6,1,-6,2,4,-5,4,-1,2,1,0,3,0,2,3,-5,1,-2
1	2,3,1,-5,2,-6,2,0,2,1,0,3,0
2	5,2,1,-6,3,-6,1,0,-5,2,4,-5,4,-4,2,0,0,4,0,3,0,-1,2,-3,3,-3
3	5,2,0,-6,4,-6,3,4,-5,2,-3,3,-3,2,4,-2,4,-1,2,1,0,3,0,1,0,-1
4	1,5,3,0,3,-6,0,-3,0,-2,4,-2
5	4,4,4,-6,0,-6,0,-4,3,-4,2,4,-3,4,-1,2,1,0,3,0,1,0,-1
6	1,5,0,-6,0,0,4,0,4,-3,0,-3
7	1,5,0,-6,4,-6,4,-5,1,-2,1,0
8	7,2,1,0,3,0,2,1,-3,3,-3,2,1,-6,3,-6,2,0,-5,0,-4,2,0,-1,0,-2,2,4,-5,4,-4,2,4,-1,4,-2
9	1,5,4,0,4,-6,0,-6,0,-3,4,-3
:	2,1,2,-2,1,2,-4
;	3,1,2,-3,1,1,1,2,2,0,2,-1
<	1,3,3,-6,0,-3,3,0
=	2,2,0,-2,4,-2,2,0,-4,4,-4
>	1,3,1,-6,4,-3,1,0
?	4,1,2,0,3,2,-2,2,-3,4,-5,2,1,-6,3,-6,1,0,-5
@	0
A	2,5,0,0,0,-4,2,-6,4,-4,4,0,2,0,-2,4,-2
B	4,4,3,0,0,0,0,-6,3,-6,2,0,-3,3,-3,2,4,-1,4,-2,2,4,-4,4,-5
C	5,2,0,-1,0,-5,2,1,-6,3,-6,2,1,0,3,0,1,4,-5,1,4,-1
D	2,4,3,0,0,0,0,-6,3,-6,2,4,-5,4,-1
E	2,4,4,0,0,0,0,-6,4,-6,2,0,-3,3,-3
F	2,3,0,0,0,-6,4,-6,2,0,-3,3,-3
G	5,2,0,-1,0,-5,2,1,0,3,0,2,1,-6,3,-6,1,4,-5,3,4,-1,4,-2,3,-2
H	3,2,0,0,0,-6,2,4,0,4,-6,2,0,-3,4,-3
I	3,2,1,0,3,0,2,1,-6,3,-6,2,2,0,2,-6
J	3,2,4,-6,4,-1,2,1,0,3,0,1,0,-1
K	2,2,0,0,0,-6,3,4,-6,1,-3,4,0
L	1,3,0,-6,0,0,4,0
M	1,5,0,0,0,-6,2,-3,4,-6,4,0
N	3,2,0,0,0,-6,2,4,0,4,-6,2,1,-4,3,-2
O	4,2,1,0,3,0,2,1,-6,3,-6,2,0,-1,0,-5,2,4,-1,4,-5
P	3,3,0,0,0,-6,3,-6,2,4,-5,4,-4,2,0,-3,3,-3
Q	5,2,0,-1,0,-5,2,1,-6,3,-6,2,4,-5,4,-1,2,4,0,2,-2,2,1,0,2,0
R	4,3,0,0,0,-6,3,-6,2,4,-5,4,-4,2,3,-3,0,-3,2,2,-3,4,0
S	7,1,0,-1,2,1,0,3,0,2,4,-1,4,-2,2,1,-3,3,-3,2,0,-4,0,-5,2,1,-6,3,-6,1,4,-5

```
T  2,2,2,0,2,-6,2,0,-6,4,-6
U  3,2,0,-6,0,-1,2,4,-1,4,-6,2,1,0,3,0
V  1,5,0,-6,0,-2,2,0,4,-2,4,-6
W  5,2,0,-6,0,0,1,1,-1,2,2,-2,2,-3,1,3,-1,2,4,0,4,-6
X  2,2,0,-6,4,0,2,4,-6,0,0
Y  2,3,2,0,2,-4,0,-6,2,4,-6,3,-4
Z  3,2,0,0,4,0,2,0,-6;4,-6,2,0,-1,4,-5
[  2,4,4,0,0,0,0,-6,4,-6,2,1,0,1,-6
\  1,2,0,-5,4,-1
]  2,4,0,0,4,0,4,-6,0,-6,2,3,-6,3,0
^  1,3,0,-3,2,-5,4,-3
_  1,2,0,0,4,0
```

5.3 A SIMPLE GRAPHING PROGRAM

You can draw graphs, with axes, tick marks, frames, grids, and labels, using the PLOT package. It begins by pausing briefly for some initialization (actually reading a text file of data that defines the font to be used for the captions) and then displays the main menu:

```
             ***MAIN MENU OPTIONS***
      1.  SET DISPLAY OPTIONS
      2.  INPUT DATA VALUES
      3.  PLOT GRAPH
      4.  QUIT
          WHICH OPTION:
```

Respond by typing 1. A secondary menu then appears, with the current values of the parameters and options that you can change if you wish. These are initialized to convenient default values.

```
                      OPTIONS
         OPTION                        VALUE
      1. SET X-MIN AND X-MAX          -100,100
      2. SET Y-MIN AND Y-MAX          -100,100
      3. SET AXES                     CROSSED
      4. SET TICK MARKS(PLACEMENT)    BOTH AXES
      5. SET TICK MARKS(X-AXIS)       10
      6. SET TICK MARKS(Y-AXIS)       10
      7. GRID                         NULL
      8. FRAME                        NULL
      9. CAPTION FOR  X-AXIS          X-AXIS
      10. CAPTION FOR  Y-AXIS         Y-AXIS
      11. TITLE FOR GRAPH             A GRAPH
      12. RETURN TO MAIN MENU
          WHICH OPTION?
```

You can type 12 to return to the main menu directly, without making any changes, or you can change one or more of the options, in any order. After you type an option number, a susidiary prompt appears, and you respond as follows:

1. SET X-MIN AND X-MAX: Type the minimum and maximum x values of the data to be plotted. These are expressed in user coordinates.

2. SET Y-MIN AND Y-MAX: Type the minimum and maximum *y* values of the data, in user coordinates. The *y* values increase upward.
3. SET AXES: Type the letter that determines the position of the axes as follows: C to cross them at the position $x=0$, $y=0$; P to display them at the bottom and left of the screen; S to suppress them.
4. SET TICK MARKS (PLACEMENT): Type the letter X, Y, B, or S, to provide tick marks on the *x* axis, on the *y* axis, on both axes, or to suppress them, respectively.
5. SET TICK MARKS (X-AXIS): Type the interval, measured in user coordinates, for the tick marks on the *x* axis.
6. SET TICK MARKS (Y-AXIS): Type the corresponding interval for the *y* axis.
7. SET GRID: Type D or N to display or omit a grid through the tick marks.
8. SET FRAME: Type D to display a frame around the plotting area, or N to omit it.
9. CAPTION FOR X-AXIS: Type the caption.
10. CAPTION FOR Y-AXIS: Type the caption.
11. TITLE OF GRAPH: Type the title.

You type 12 to return to the main menu, when the options and parameters are set appropriately. Then you type 2, to input the data to be plotted. The program prompts for the number of points, and then for the coordinates of these. If you type an *x* or *y* value that is outside the limits set by option 1 or 2, the program displays a flashing message that asks you to enter valid data.

When you have typed the coordinates of the appropriate number of points, the main menu reappears on the screen. You type 3 to plot the data that you just entered. After a short delay, during which another program is loaded, the main menu appears again. You type 3 once more, and the program draws the axes, tick marks, tick mark labels, frame, and grid (or whichever subset of these has been requested). It then draws straight lines between the successive points that you specified, in the order in which their coordinates were entered.

Figures 5.4A and 5.4B illustrate the main provisions of the program.

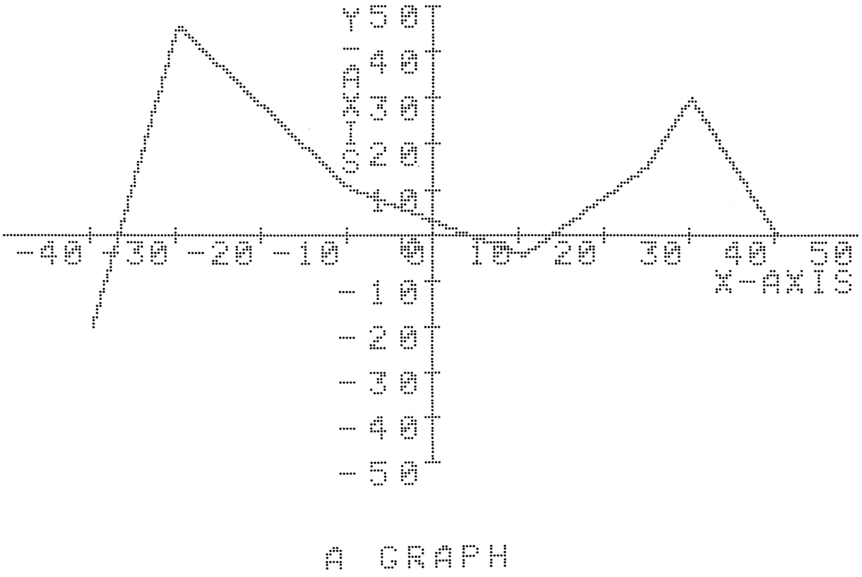


Figure 5.4A

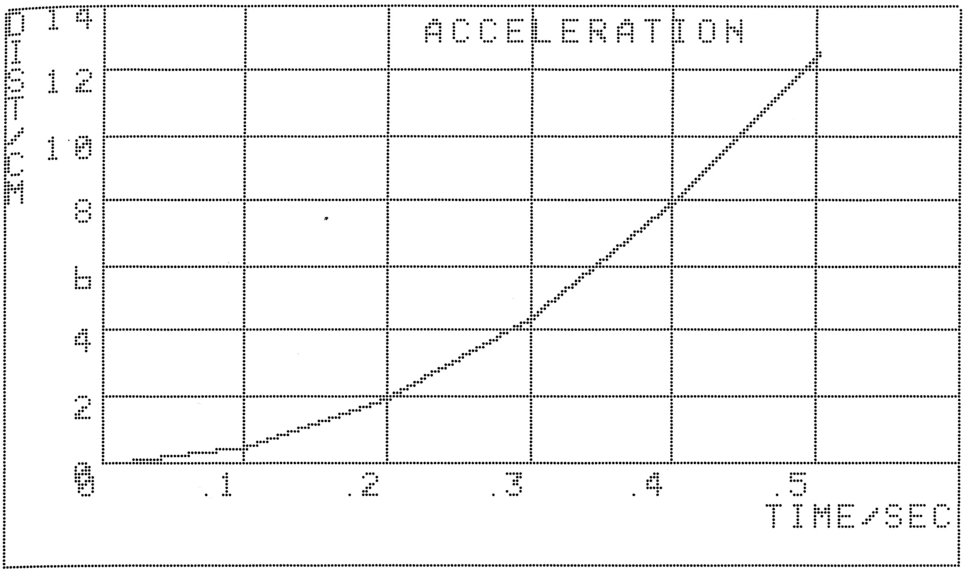


Figure 5.4B

The main variables in the program are as follows

`X0,X9`	The limits on *x* in user coordinates
`Y0,Y9`	The limits on *y* in user coordinates
`AX$`	The option that controls the position of the axes
`TX,TY`	The tick mark intervals on the *x* and *y* axes
`TM$`	The option that controls the position of tick marks
`FR$`	The option that controls whether a graph is framed
`XC$,YC$`	The captions on the *x* and *y* axes
`TI$`	The title of the graph
`MM$`	A main menu option
`XL,XR`	The limits on *x* in screen coordinates
`YL,YR`	The limits on *y* in screen coordinates
`PT%(300,2)`	The coordinates that define the font data. These are explained in Section 5.2
`DS%(200)`	The directory of the PT% array (see Section 5.2)
`DC%(70)`	The directory of the DS% array (see Section 5.2)
`NC,NS,NP`	The limits on the loops that input the labelling font
`CN,SN,PN`	The indexes of these loops
`OP`	The number of the option selected from the main menu
`ND`	The number of points to be plotted on a graph
`XD,YD`	The coordinates of a point to be plotted
`XU,XS`	The tick mark values on the *x* axis
`YU,YS`	The tick mark values on the *y* axis
`OX,OY`	The screen coordinates of the start of a line to be plotted
`NX,NY`	The coordinates at the end of the line

The PLOT package actually consists of three programs—PLOT1, PLOT2, and PLOT3. These are switched mechanically whenever necessary—they occupy too much space to all fit in the memory of the computer at the same time. The chaining feature of the disk operating system is used to pass information between the programs and to allow the automatic switching. The CHAIN program must be copied from the system master disk to the disk where you save the PLOT package. The way to do this is described in the Apple DOS Manual (reference [4], p. 183; see beginning of Chapter 8).

The PLOT1 program reads the text file of font data that is used to draw labels and captions. The arrays and other variables, the underlying logic, and most of the actual statements are taken from the FONT FORMER program, which is discussed in the preceding section. PLOT1 also performs some further initialization, and then loads PLOT2. This displays the main menu, and deals with its options 1 and 2, to set the display options and to input the data for plotting. When you request option 3 of the main menu, to plot a graph, the program PLOT2 replaces itself by PLOT3. This redisplays the main menu, and you type 3 again. The screen clears and the graph is drawn. It stays on the screen until you press RETURN (or any other key), to bring back the main menu. To plot new data, without any change in the plotting options, you just type 2. The program prompts for the number and coordinates of points just as before, and the graph is displayed by requesting option 3 at the end. PLOT3 handles this by a duplicate of the small portion of PLOT2 that deals with input. If you type 1, however, to change the plotting options, PLOT3 replaces itself by PLOT2.

Because of storage constraints, the graphs are plotted in the HGR2 mode. This lets some variables occupy the region of memory that maps the screen in the HGR mode.

Suggested variations

1. Allow points to be plotted with or without connecting lines.
2. Allow the data points to be marked by special symbols.
3. Allow the texture of the lines to be varied (for example, to be either solid or to consist of dots, or dashes, or dots and dashes in alternation).
4. Let *y* values increase either downward or upward.
5. Allow the area between the graph and one of the axes to be shaded by the program.
6. Allow multiple plots in the same display.
7. Let the user specify the width, height, and position of a plotting area that does not occupy the entire screen.
8. Allow multiple plots of this kind on the same screen.

The PLOT1 program

```
100  REM *** PLOT1-FOR MAKING GRAPHS ***
110  HIMEM: 16384
120  DIM TI$(12)
130  HGR2 : TEXT
140  FOR LO=1 TO 12: READ TI$(LO): NEXT LO
150  DATA SET X-MIN AND X-MAX,SET Y-MIN AND Y-MAX,SET AXES
```

```
 160  DATA SET TICK MARKS(PLACEMENT),SET TICK MARKS(X-AXIS)
 170  DATA SET TICK MARKS(Y-AXIS)
 180  DATA GRID, FRAME,CAPTION FOR X-AXIS,CAPTION FOR Y-AXIS,TITLE FOR GRAPH
 190  DATA RETURN TO MAIN MENU
 200  X0=-50:X9=50:Y0=-50:Y9=50
 210  AX$="CROSSED":TM=10:TM$="BOTH AXES":TY=10
 220  ID$="NULL":FR$="NULL":XC$="X-AXIS"
 230  YC$="Y-AXIS":TI$="A GRAPH"
 240  DIM XD(20),YD(20)
 250  FOR LO=1 TO 4: READ MM$(LO): NEXT
 260  DATA SET DISPLAY OPTIONS,INPUT DATA VALUES,PLOT GRAPH,QUIT
 270  XL=0:XR=279:YU=0:YL=159
 280  TEXT : HOME : PRINT "READING IN LABEL DATA"
 290  GOSUB 1000
 300  PRINT CHR$(4);"BLOAD CHAIN, A520": REM *** CALL PLOT2 ***
 310  CALL 520"PLOT2"
1000  REM *** READ IN DATA FOR LABELS FROM A TEXT FILE ***
1010  DIM DC%(70),DS%(200),PT%(360,2)
1020  PRINT CHR$(4);"OPEN LABELTEXT"
1030  PRINT CHR$(4);"READ LABELTEXT"
1040  INPUT NC: REM *** READS NO. OF CHARACTERS ***
1050  DC%(0)=0:DS%(0)=0
1060  FOR CN=1 TO NC
1070  INPUT NS:DC%(CN)=DC%(CN-1)+NS
1080  IF NS=0 THEN 1140
1090  FOR SN=DC%(CN-1)+1 TO DC%(CN)
1100  INPUT NP:DS%(SN)=DS%(SN-1)+NP
1110  FOR PN=DS%(SN-1)+1 TO DS%(SN)
1120  INPUT PT%(PN,1): INPUT PT%(PN,2)
1130  NEXT PN,SN
1140  NEXT CN
1150  PRINT CHR$(4);"CLOSE LABELTEXT"
1160  RETURN
```

The PLOT2 program

```
 100  REM *** PLOT2 ***LIST
 500  REM *** MAIN MENU ***
 510  TEXT : HOME : INVERSE : HTAB 12: PRINT "***MAIN MENU OPTIONS***"
 520  NORMAL
 530  VTAB 4: FOR LO=1 TO 4: HTAB 6: PRINT LO;".  ";MM$(LO): NEXT LOP
 540  PRINT : PRINT : PRINT
 550  HTAB 10: INPUT "WHICH OPTION";OP
 560  ON OP GOTO 1000,2000,3000,580
 570  VTAB PEEK(37)-1: GOTO 550
 580  END
1000  HCOLOR=0: HPLOT 0,0: CALL 62454
1010  TEXT : HOME : INVERSE : HTAB 16: PRINT "OPTIONS"
1020  VTAB 3: HTAB 4: PRINT "OPTION";: HTAB 34: PRINT "VALUE"
```

```
1030  VTAB 6: NORMAL
1040  FOR LO=1 TO 11: PRINT LO;".  ";TI$(LO): NEXT LO
1050  POKE 34,18: INVERSE
1060  VTAB 6: HTAB 31: PRINT X0;",";X9,: HTAB 31: PRINT Y0;",";Y9,
1070  HTAB 31: PRINT AX$,: HTAB 31: PRINT TM$,
1080  HTAB 31: PRINT TM: HTAB 31: PRINT TY
1090  HTAB 31: PRINT ID$: HTAB 31: PRINT FR$: HTAB 31
1100  PRINT XC$: HTAB 31: PRINT YC$: HTAB 31: PRINT LEFT$(TI$,10)
1110  PRINT "12.  ";TI$(12)
1120  HOME : PRINT : PRINT : PRINT
1130  NORMAL : HTAB 5
1140  VTAB 20: INPUT "WHICH OPTION? ";OP%
1150  HOME : PRINT
1160  ON OP% GOTO 1180,1220,1290,1420,1390,1260,1530,1600,1670,1710,1750,500
1170  PRINT "NON-EXISTENT OPTION": GOTO 1130
1180  REM *** RESET X MAXIMUM AND X-MINIMUM ***
1190  INPUT "NEW X-MIN AND X-MAX VALUES ARE ";X0,X9
1200  IF X0>X9 THEN XI=X0:X0=X9:X9=XI
1210  GOTO 1010
1220  REM *** RESET Y MAXIMUM AND Y MINIMUM ***
1230  INPUT "NEW Y-MIN AND Y-MAX VALUES ARE ";Y0,Y9
1240  IF Y0>Y9 THEN YI=Y0:Y0=Y9:Y9=YI
1250  GOTO 1010
1260  REM *** SET Y-AXIS TICK INTERVAL ***
1270  INPUT "NEW Y-AXIS TICKMARK INTERVAL IS ";TY:TY=ABS(TY)
1280  GOTO 1010
1290  REM *** SET AXES ***
1300  PRINT "YOU MAY EITHER :"
1310  PRINT "(S)UPPRESS THE AXES": PRINT "(C)ENTER THEM ON THE SCREEN, OR"
1320  PRINT "(P)LACE THEM AT THE BOTTOM LEFT"
1330  INPUT "WHICH ONE? ";IN$
1340  IF LEFT$(IN$,1)="S" THEN AX$="SUPPRESSED": GOTO 1380
1350  IF LEFT$(IN$,1)="C" THEN AX$="CROSSED": GOTO 1380
1360  IF LEFT$(IN$,1)="P" THEN AX$="BOTTOM": GOTO 1380
1370  PRINT CHR$(7): GOTO 1290
1380  GOTO 1010
1390  REM *** SET TICK MARK INTERVAL ***
1400  INPUT "NEW X-AXIS TICK MARK INTERVAL IS ";TM:TM=ABS(TM)
1410  GOTO 1010
1420  REM SET TICK MARK PLACEMENT
1430  PRINT "YOU MAY DISPLAY TICKS ON EITHER: "
1440  PRINT "(X)-AXIS ONLY": PRINT "(Y)-AXIS ONLY": PRINT "(B)OTH AXES"
1450  PRINT "(S)UPPRESSED"
1460  INPUT "WHICH OPTION? ";IN$
1470  IF LEFT$(IN$,1)="X" THEN TM$="X-AXIS": GOTO 1520
1480  IF LEFT$(IN$,1)="Y" THEN TM$="Y-AXIS": GOTO 1520
1490  IF LEFT$(IN$,1)="B" THEN TM$="BOTH AXES": GOTO 1520
1500  IF LEFT$(IN$,1)="S" THEN TM$="SUPPRESSED": GOTO 1520
1510  GOTO 1420
```

```
1520  GOTO 1010
1530  REM *** SET GRID ***
1540  PRINT "GRID IS SET TO ";ID$:
1550  INPUT "NEW GRID SETTING IS(NULL OR DISPLAY)";IN$
1560  IF LEFT$(IN$,1)="N" THEN ID$="NULL": GOTO 1590
1570  IF LEFT$(IN$,1)="D" THEN ID$="DISPLAY": GOTO 1590
1580  GOTO 1540
1590  GOTO 1010
1600  REM *** SET FRAME ***
1610  PRINT "FRAME IS SET TO ";FR$
1620  INPUT "NEW FRAME SETTING IS(NULL OR DISPLAY)";IN$
1630  IF LEFT$(IN$,1)="N" THEN FR$="NULL": GOTO 1660
1640  IF LEFT$(IN$,1)="D" THEN FR$="DISPLAY": GOTO 1660
1650  GOTO 1610
1660  GOTO 1010
1670  REM *** SET CAPTION FOR X AXIS ***
1680  PRINT "CAPTION SHOULD READ: ": HTAB 8: INPUT "";XC$
1690  XC$=LEFT$(XC$,8)
1700  GOTO 1010
1710  REM *** SET Y CAPTION ***
1720  PRINT "CAPTION SHOULD READ: ": HTAB 8: INPUT "";YC$
1730  YC$=LEFT$(YC$,8)
1740  GOTO 1010
1750  REM *** INPUT TITLE FOR GRAPH ***
1760  PRINT "TITLE SHOULD READ": HTAB 10: INPUT "";TI$
1770  TI$=LEFT$(TI$,14)
1780  GOTO 1010
2000  REM *** COLLECT DATA VALUES ***
2010  TEXT : HOME : HTAB 10: INVERSE : PRINT "SUBROUTINE TO COLLECT DATA"
2020  VTAB 4
2030  INPUT "HOW MANY DATA VALUES(0 FOR MAIN MENU)";ND
2040  IF ND>20 THEN  FLASH : GOTO 2130
2050  IF ND=0 THEN 500
2060  NORMAL
2070  FOR IN=1 TO ND
2080  PRINT "PLEASE ENTER DATA PAIR #";IN;: INPUT " ";XD,YD
2090  IF XD<X0 OR YD<Y0 OR XD>X9 OR YD>Y9 THEN 2120
2100  XD(IN)=XD:YD(IN)=YD
2110  NEXT IN: GOTO 500
2120  FLASH : PRINT "ILLEGAL VALUE": NORMAL : VTAB PEEK(37)-1: GOTO 2080
2130  PRINT "NO MORE THAN 20, PLEASE": INVERSE : GOTO 2030
3000  PRINT CHR$(4);"BLOAD CHAIN, A520"
3010  PRINT "LOADING PLOT3"
3020  CALL 520"PLOT3"
```

The PLOT3 program

```
 100  REM *** PLOT3 ***
 500  REM *** MAIN MENU ***
 510  TEXT : HOME : INVERSE : HTAB 12: PRINT "***MAIN MENU OPTIONS***"
 520  NORMAL
 530  VTAB 4: FOR LO=1 TO 4: HTAB 6: PRINT LO;".  ";MM$(LO): NEXT LOP
 540  PRINT : PRINT : PRINT
 550  HTAB 10: INPUT "WHICH OPTION";OP
 560  ON OP GOTO 1000,2000,3000,580
 570  VTAB PEEK(37)-1: GOTO 550
 580  END
1000  PRINT CHR$(4);"BLOAD CHAIN, A520"
1010  HOME : HTAB 13: PRINT "LOADING PLOT2"
1020  CALL 520"PLOT2"
2000  REM *** COLLECT DATA VALUES ***
2010  TEXT : HOME : HTAB 10: INVERSE : PRINT "SUBROUTINE TO COLLECT DATA"
2020  VTAB 4
2030  INPUT "HOW MANY DATA VALUES(0 FOR MAIN MENU)";ND
2040  IF ND>20 THEN  FLASH : GOTO 2130
2050  IF ND=0 THEN 500
2060  NORMAL
2070  FOR IN=1 TO ND
2080  PRINT "PLEASE ENTER DATA PAIR #";IN;: INPUT " ";XD,YD
2090  IF XD<X0 OR YD<Y0 OR XD>X9 OR YD>Y9 THEN 2120
2100  XD(IN)=XD:YD(IN)=YD
2110  NEXT IN: GOTO 500
2120  FLASH : PRINT "ILLEGAL VALUE": NORMAL : VTAB PEEK(37)-1: GOTO 2080
2130  PRINT "NO MORE THAN 20, PLEASE": INVERSE : GOTO 2030
3000  REM *** PLOT GRAPHS ***
3010  POKE -16297,0: POKE -16302,0: POKE -16304,0: POKE -16299,0
3020  HCOLOR=3
3030  XS=250/(X9-X0)
3040  YS=130/(Y9-Y0)
3050  IF LEFT$(AX$,1)="S" THEN 3810
3060  IF LEFT$(AX$,1)="C" THEN 3500
3070  IF LEFT$(FR$,1)="N" THEN 3090
3080  HPLOT XL,YU TO XR,YU TO XR,YL TO XL,YL TO XL,YU
3090  HPLOT 29,0 TO 29,130 TO 279,129
3100  IF LEFT$(TM$,1)="S" THEN 3230
3110  IF LEFT$(TM$,1)="Y" THEN 3160
3120  FOR X=29 TO 279 STEP XS*TM
3130  HPLOT X,130 TO X,127:XV=X:YV=139:V=X0+(X-29)/XS
3140  GOSUB 5500: NEXT X
3150  IF LEFT$(TM$,1)="X" THEN 3230
3160  FOR Y=130 TO 0 STEP -YS*TY
3170  HPLOT 30,Y TO 33,Y
3180  V=Y9-Y/YS: IF ABS(V)<.003 THEN V=0
3190  V$=STR$(V):XV=30-9*LEN(V$)
3200  YV=Y+6: FOR LA=1 TO LEN(V$)
3210  C$=MID$(V$,LA,1): GOSUB 5750
3220  XV=XV+9: NEXT LA,Y
3230  GOSUB 5000
```

```
3240  GOSUB 4500
3250  IF LEFT$(ID$,1)="N" THEN 3320
3260  FOR X=29 TO 279 STEP XS*TM
3270  HPLOT X,0 TO X,130
3280  NEXT X
3290  FOR Y=130 TO 0 STEP -YS*TY
3300  HPLOT 30,Y TO 279,Y
3310  NEXT Y
3320  XV%=1:YV%=148: GOSUB 5580
3330  GET A$
3340  GOTO 500
3500  REM *** AXES CROSSED ***
3510  C1=YS*Y9:C2=279-XS*X9
3520  HPLOT 29,C1 TO 279,C1
3530  HPLOT C2,0 TO C2,130
3540  IF LEFT$(TM$,1)="Y" THEN 3670
3550  IF LEFT$(TM$,1)="S" THEN 3810
3560  FOR X=C2 TO 279 STEP XS*TM
3570  HPLOT X,C1-2 TO X,C1+2
3580  YV=C1+8:XV=X:V=X0+(X-29)/XS
3590  GOSUB 5500
3600  NEXT X
3610  FOR X=C2 TO 30 STEP -TM*XS
3620  HPLOT X,C1-2 TO X,C1+2
3630  YV=C1+8:XV=X:V=X0+(X-29)/XS
3640  GOSUB 5500
3650  NEXT X
3660  IF LEFT$(TM$,1)="X" THEN 3810
3670  FOR Y=C1 TO 0 STEP -TY*YS
3680  HPLOT C2-2,Y TO C2+2,Y
3690  V=Y9-Y/YS:V$=STR$(V):XV=C2-9*LEN(V$)
3700  YV=Y+6: FOR LA=1 TO LEN(V$)
3710  C$=MID$(V$,LA,1): GOSUB 5750
3720  XV=XV+9
3730  NEXT LA,Y
3740  FOR Y=C1 TO 130 STEP TY*YS
3750  HPLOT C2-2,Y TO C2+2,Y
3760  V=Y9-Y/YS:V$=STR$(V):XV=C2-9*LEN(V$)
3770  YV=Y+6: FOR LA=1 TO LEN(V$)
3780  C$=MID$(V$,LA,1): GOSUB 5750
3790  XV=XV+9
3800  NEXT LA,Y
3810  IF LEFT$(FR$,1)="N" THEN 3830
3820  HPLOT 279,0 TO 279,130 TO 30,130 TO 30,0 TO 279,0
3830  GOSUB 5000
3840  GOSUB 4500
3850  IF LEFT$(ID$,1)="N" THEN 4010
3860  FOR X=C2 TO 279 STEP TM*XS
3870  HPLOT X,0 TO X,130
```

```
3880  NEXT X
3890  FOR X=C2 TO 30 STEP -XS*TM
3900  HPLOT X,0 TO X,130
3910  NEXT X
3920  FOR Y=C1 TO 0 STEP -TY*YS
3930  HPLOT 30,Y TO 279,Y
3940  NEXT Y
3950  FOR Y=C1 TO 130 STEP TY*YS
3960  HPLOT 30,Y TO 279,Y
3970  NEXT Y
3980  FOR X=C2-2 TO C2+2
3990  HPLOT X,C1-2 TO X,C1+2
4000  NEXT
4010  XV%=C2-26:YV%=C1+16
4020  GOSUB 5580
4030  GET A$: GOTO 500
4500  REM *** DRAW TITLE OF GRAPH ***
4510  XV=115
4520  YV=10
4530  IF LEFT$(AX$,1)="C" THEN YV=160
4540  FOR LO=1 TO LEN(TI$):XV=XV+8:C$=MID$(TI$,LO,1)
4550  GOSUB 5750
4560  NEXT LO
4570  RETURN
4750  END
5000  OX=30+(XD(1)-X0)*XS
5010  OY=-(YD(1)-Y0)*YS+130
5020  FOR NP=2 TO ND
5030  NX=30+(XD(NP)-X0)*XS
5040  NY=130-(YD(NP)-Y0)*YS
5050  HPLOT OX,OY TO NX,NY
5060  OX=NX:OY=NY
5070  NEXT NP
5080  RETURN
5500  V$=STR$(V)
5510  XV=XV-7*LEN(V$)
5520  FOR LA=1 TO LEN(V$)
5530  C$=MID$(V$,LA,1)
5540  GOSUB 5750
5550  XV=XV+7
5560  NEXT LA
5570  RETURN
5580  XV=XV%:YV=7
5590  FOR LA=1 TO LEN(YC$):C$=MID$(YC$,LA,1)
5600  GOSUB 5750
5610  YV=YV+8
5620  NEXT LA
5630  XV=279-7*LEN(XC$):YV=YV%
5640  FOR LA=1 TO LEN(XC$)
```

```
5650  C$=MID$(XC$,LA,1)
5660  GOSUB 5750
5670  XV=XV+7
5680  NEXT LA
5690  RETURN
5750  CN=ASC(C$)-31
5760  S1=DC%(CN-1)+1:S2=DC%(CN)
5770  IF S1>S2 THEN  RETURN
5780  FOR SN=S1 TO S2
5790  P1=DS%(SN-1)+1:P2=DS%(SN)-1
5800  IF P2>=P1 THEN 5830
5810  HPLOT XV+PT%(P1,1),YV+PT%(P1,2)
5820  GOTO 5860
5830  FOR PN=P1 TO P2
5840  HPLOT XV+PT%(PN,1),YV+PT%(PN,2) TO XV+PT%(PN+1,1),YV+PT%(PN+1,2)
5850  NEXT PN
5860  NEXT SN
5870  RETURN
```

5.4 BARS AND BAR CHARTS

Bar charts provide very effective summaries of many kinds of business and other data. The program BAR CHART plots bar charts and histograms. It first displays the main menu, with the default values for some of the options.

```
               *** MAIN MENU ***
         OPTION                      VALUE
      1. DIRECTION OF BARS           VERTICAL
      2. WIDTH OF BARS               15
      3. GAP BETWEEN BARS            10
      4. AXIS DATA                   100,20,5
      5. INPUT TITLE
      6. INPUT DATA VALUES
      7. QUIT
       PLEASE TYPE OPTION:
```

If you accept the defaults, and want a chart like Figure 5.5, just type 6. The menu disappears and the axes appear in the graphics area of the screen. Beneath this, in the text area, the program prompts for the number of data items. After you have typed this, the program prompts for the height and label of each of the bars in turn. A bar appears as soon as you type the data for it. When the chart is complete, it stays on the screen until you press any key. The screen clears and the main menu reappears for you to create another chart or quit.

You can vary the style of the chart by using one or more of the options 1–4. After you type an option number, a subsidiary prompt appears and you respond as follows:

1. TYPE 1 FOR VERTICAL, 2 FOR HORIZONTAL: Type 1 or 2, to specify the directions of the bars.
2. BAR WIDTH: Type the width in screen units (pixels).

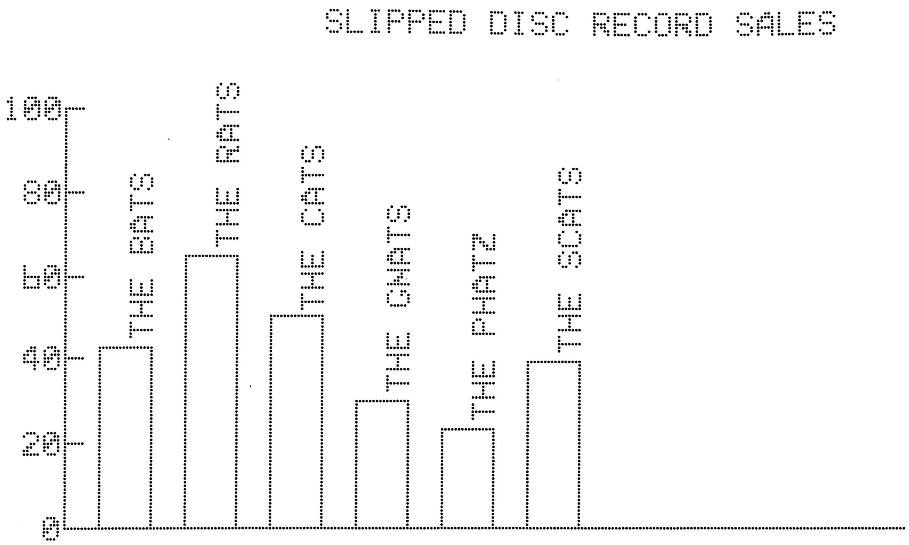


Figure 5.5

3. GAP BETWEEN BARS: Type the gap in screen units (pixels).
4. MAXIMUM VALUE REPRESENTED: Type the maximum height or length, in user units, for vertical and horizontal bars, respectively.
   TICK MARK INTERVAL: Type the interval in screen units (pixels).
   MINIMUM CLEARANCE FOR CAPTIONS: Type the minimum number of letters for which space must be left to label the bars.

The BAR CHART options thus let you produce bar charts such as those shown in Figures 5.6A and 5.6B. Options 3 and 4 were used for the first, to reduce the gap between the bars to zero, and to change the limit on the vertical axis. Option 1 was used for the second, to make the bars horizontal.

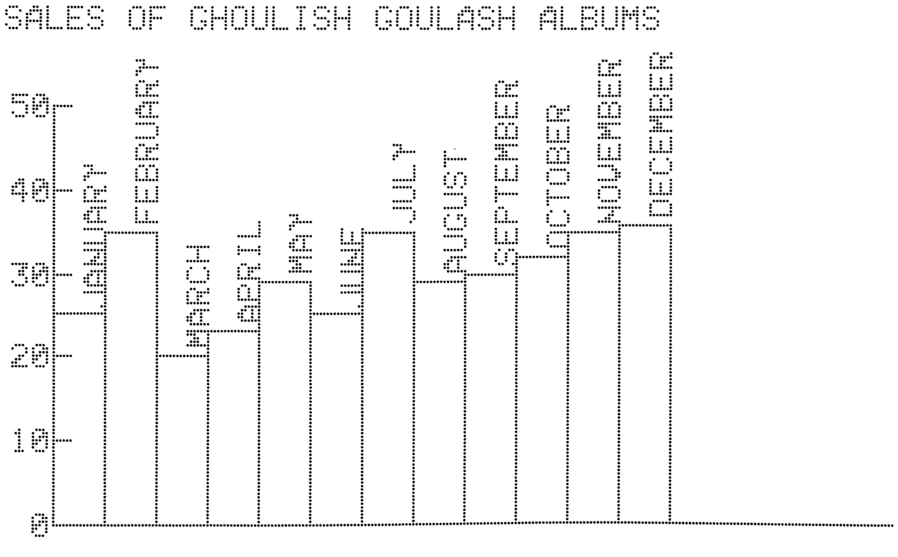


Figure 5.6A

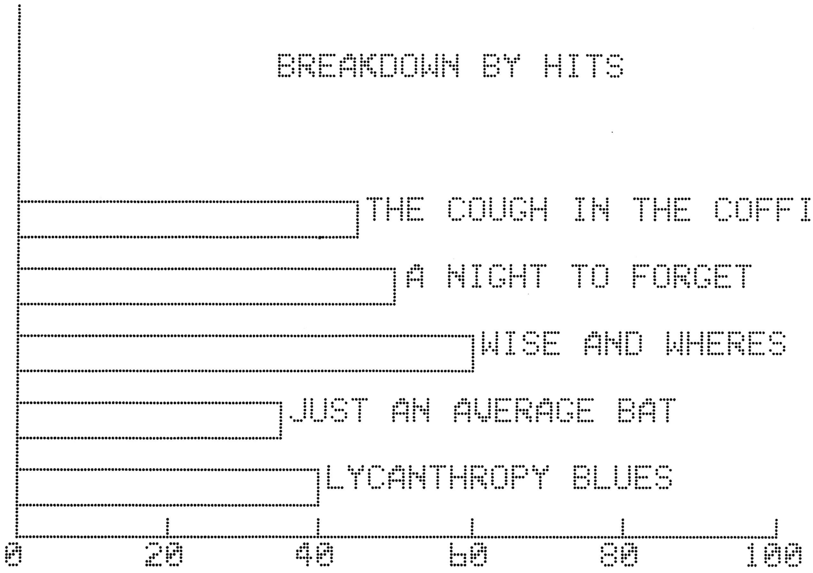


Figure 5.6B

The main variables in the program are as follows

`PT%(300,2)`	The coordinates that define the font data. These are explained in Section 5.2
`DS%(200)`	The directory of the PT% array (see Section 5.2)
`DC%(70)`	The directory of the DS% array (see Section 5.2)
`BD`	The code for the direction of the bars—1 for vertical and 2 for horizontal, respectively
`BW`	The width of the bars
`BG`	The gap between bars
`MV`	The maximum value of the user variable marked on the axis
`TI`	The tick mark interval on the axis
`H`	The height of the letters used in the title and the captions
`W`	The width of the letters
`MC%`	The minimum number of letters for which space must be left above vertical bars and at the right of horizontal bars
`BD$(2)`	The word VERTICAL or HORIZONTAL that the user employs to specify the direction of the bars
`OP`	The option selected from the main menu
`XO,YO`	The position of the origin of the chart on the screen
`DE`	The index of a loop that provides a delay
`TT$`	The title of the chart
`XT,YT`	The position of the beginning of the title on the screen
`A$`	The dummy variable in the statement that waits for a key to be pressed
`BC`	The distance between the left edges of successive vertical bars or between the lower edges of successive horizontal bars

SC	The scale factor to convert values in user coordinates to heights or lengths of bars in screen coordinates—this maps the range 0–MV in user coordinates onto a portion of the screen that allows MC% characters of width W above or alongside the bars, and 3 screen units clearance
V	The position of a tick mark along the axis in user coordinates
X,Y	The *x* or *y* position of the tick mark in screen coordinates
X1	The *x* value of the left edge of a vertical bar
Y1	The *y* value of the lower edge of a horizontal bar
VA	The value represented by the height or length of a bar
CA$	The caption on a bar
XL,YL	The position at which the caption begins on the screen
ST$	The string to be displayed by the string plotting subroutine
M1,M2	The directions in which letters are drawn and strings are written (see FONT FORMER in Section 5.2)
LL	The number of characters in a tick mark label
MC	The maximum length of a caption that can fit above or alongside a bar

The portions of the program that read the font data and display strings of characters correspond very closely to portions of FONT FORMER and use the same variables. These are explained in Section 5.2. The further details are explained by the list of variables in this section and the REM statements in the program.

Suggested variations

1. Give the user the options of vertical bars that go up or down, or horizontal bars that go right or left (Figure 5.7).

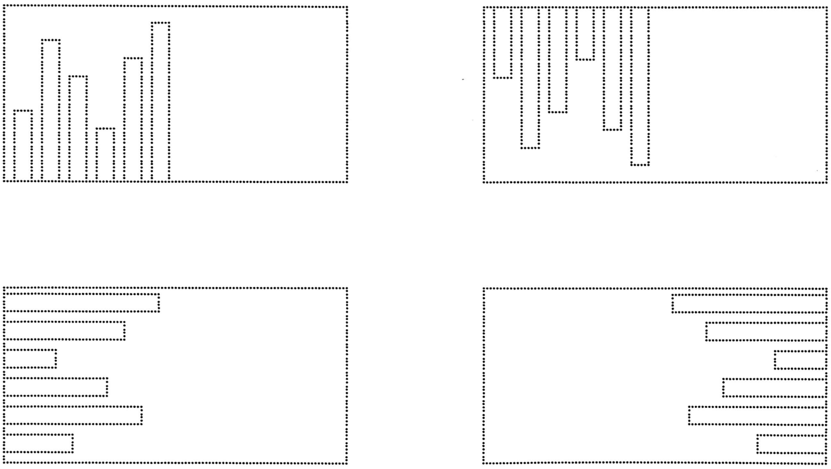


Figure 5.7

2. Let the user specify shading to be (a) solid, (b) absent, (c) with horizontal lines, (d) with vertical lines, (e) with slanting lines, (f) cross hatched, or (g) dotted.
3. Let the user choose which end of the bars is labeled.
4. Allow labeled tick marks on the axis.
5. Allow a frame.
6. Allow lines through the tick marks, across the chart.
7. Provide the three-dimensional effect (Figure 5.8).

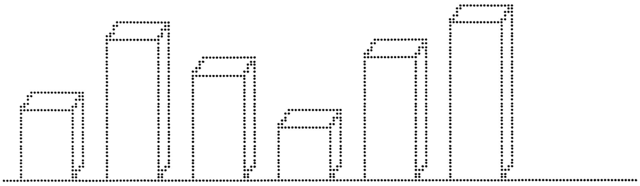


Figure 5.8

8. Let the depth of the bars represent another variable.
9. Allow a negative gap to produce the overlapping effect (Figure 5.9).

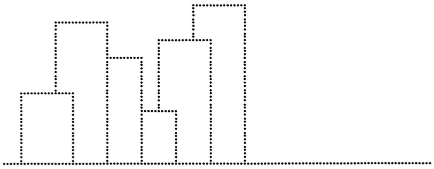


Figure 5.9

10. Allow stacking of the kind shown in Figure 5.10.

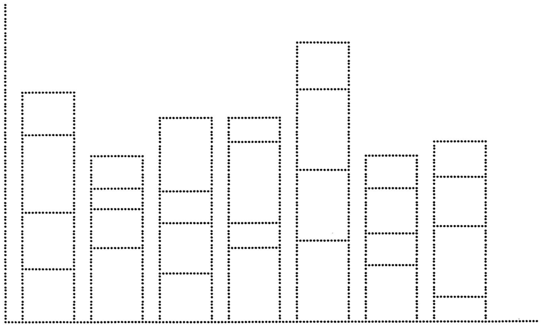


Figure 5.10

11. Let the user specify the region of the screen that the bar chart occupies.
12. Let the user provide overall captions.
13. Allow several bar charts on the screen, with separate captions.

The BAR CHART program

```
100  REM *** BAR CHART ***
110  LOMEM: 16384: REM *** PROTECT MEMORY ***
120  DIM DC%(70),DS%(200),PT%(360,2): REM *** HI-RES FONT ***
130  GOSUB 3000: REM *** LOAD HI-RES FONT ***
140  BD=1:BW=15:BG=10:MV=100:TI=20: REM *** BAR & AXIS DATA ***
150  H=8:W=6:MC%=5
160  BD$(1)="VERTICAL":BD$(2)="HORIZONTAL"
170  X0=4*W:Y0=159-H
180  TEXT : HOME
190  PRINT  TAB( 10);"*** MAIN MENU ***"
200  PRINT  TAB( 05);"OPTION"; SPC( 20);"VALUE"
210  PRINT " "
220  PRINT "1. DIRECTION OF BARS"; TAB( 30);BD$(BD)
230  PRINT "2. WIDTH OF BARS"; TAB( 30);BW
240  PRINT "3. GAP BETWEEN BARS"; TAB( 30);BG
250  PRINT "4. AXIS DATA"; TAB( 30);MV;", ";TI;", ";MC%
260  PRINT "5. INPUT TITLE"
270  PRINT "6. INPUT DATA VALUES"
280  PRINT "7. QUIT"
290  VTAB 12: HTAB 5
300  CALL -958: REM *** CLEAR REST OF SCREEN ***
310  INPUT "PLEASE TYPE OPTION ";OP
320  ON OP GOTO 360,370,380,390,420,490,1010
330  PRINT "*** INVALID OPTION ***"
340  FOR DE=1 TO 1000: NEXT DE: GOTO 290
350  REM *** RESET PARAMETER VALUES ***
360  VTAB 13: INPUT "1 FOR VERTICAL, 2 FOR HORIZONTAL: ";BD: GOTO 180
370  INPUT "BAR WIDTH ";BW: GOTO 180
380  INPUT "GAP BETWEEN BARS ";BG: GOTO 180
390  INPUT "MAXIMUM VALUE REPRESENTED ";MV
400  INPUT "TICK MARK INTERVAL ";TI
410  INPUT "MINIMUM CLEARANCE FOR CAPTIONS: ";MC%: GOTO 180
420  VTAB 21: CALL -958
430  INPUT "TITLE FOR GRAPH: ";TT$
440  INPUT "X AND Y PLACEMENT: ";XT,YT
450  IF W*LEN(TT$)+XT<279 THEN 180
460  PRINT "TITLE TOO LONG-PRESS ANY KEY";
470  GET A$: GOTO 420
480  REM *** PLOT A BAR CHART ***
490  HGR : HCOLOR=3:BC=BG+BW
500  IF BD=2 THEN 630: REM *** BRANCH FOR HORIZONTAL BARS ***
510  REM *** BARS ARE VERTICAL ***
520  SC=(Y0-3-MC%*W)/MV: REM *** SETS SCALE FACTOR ***
530  REM *** DRAW VERTICAL AXIS AND BASE LINE ***
540  HPLOT X0,Y0-SC*MV TO X0,Y0 TO 279,Y0
550  REM *** DRAW TICK MARKS ON Y AXIS ***
560  FOR V=0 TO MV STEP TI: REM *** POSITION IN USER COORDINATES ***
```

```
570  Y=YO-SC*V: REM *** POSITION IN SCREEN COORDINATES ***
580  HPLOT XO,Y TO XO+5,Y
590  GOSUB 1020
600  NEXT V: REM *** END OF Y TICK MARK CYCLE ***
610  GOTO 730: REM *** GET INPUT DATA ***
620  REM *** BARS ARE HORIZONTAL ***
630  SC=(279-XO-3-MC%*W)/MV: REM *** SETS SCALE FACTOR ***
640  REM *** DRAW VERTICAL STARTING LINE&HORIZONTAL AXIS ***
650  HPLOT XO,0 TO XO,YO TO XO+SC*MV,YO
660  REM *** DRAW TICK MARKS ON X AXIS ***
670  FOR V=0 TO MV STEP TI: REM *** POSITION IN USER COORDINATES ***
680  X=XO+SC*V: REM *** POSITION IN SCREEN COORDINATES ***
690  HPLOT X,YO TO X,YO-5
700  GOSUB 1020
710  NEXT V: REM *** END OF TICK MARK CYCLE ***
720  REM *** GET DATA TO PLOT ***
730  HOME : VTAB 22
740  INPUT "NUMBER OF BARS ";NB
750  IF NB=0 THEN 180
760  IF BD=2 THEN 870: REM *** BRANCH FOR HORIZONTAL BARS ***
770  REM *** BARS ARE VERTICAL ***
780  REM *** CYCLE ON X VALUE OF LEFT EDGE ***
790  FOR X1=XO+BG TO XO+BG+(NB-1)*BC STEP BC
800  VTAB 23: CALL -958
810  INPUT "VALUE, CAPTION: ";VA,CA$
820  REM *** PLOT A BAR ***
830  HPLOT X1,YO TO X1,YO-SC*VA TO X1+BW,YO-SC*VA TO X1+BW,YO
840  GOSUB 1100: REM *** DRAW CAPTION ***
850  NEXT X1: REM *** END OF CYCLE FOR VERTICAL BARS ***
860  GOTO 970: REM *** GET THE OVERALL TITLE ***
870  REM *** BARS ARE HORIZONTAL ***
880  REM *** CYCLE ON Y VALUE OF LOWER EDGE ***
890  FOR Y1=YO-BG TO YO-BG-(NB-1)*BC STEP -BC
900  VTAB 23: CALL -958
910  INPUT "VALUE, CAPTION: ";VA,CA$
920  REM *** PLOT A BAR ***
930  HPLOT XO,Y1 TO XO+SC*VA,Y1 TO XO+SC*VA,Y1-BW TO XO,Y1-BW
940  GOSUB 1100: REM *** DRAW CAPTION  ***
950  NEXT Y1: REM *** END OF CYCLE FOR VERTICAL BARS ***
960  REM *** DRAW THE OVERALL TITLE ***
970  ST$=TT$:XL=XT:YL=YT
980  M1=1:M2=1: GOSUB 2500
990  HOME : VTAB 22: CALL -958
1000 PRINT "PRESS ANY KEY TO RETURN TO MAIN MENU": GET A$: GOTO 180
1010 END : REM *** TERMINATE ***
1020 REM *** DRAW TICK MARK LABEL ***
1030 ST$=LEFT$(STR$(V),4): REM *** CONVERT NUMBER TO A STRING ***
1040 LL=LEN(ST$)
1050 IF BD=1 THEN XL=XO-W*LL:YL=Y+3: REM *** ALIGN UNITS DIGIT ***
```

```
1060  IF BD=2 THEN XL=X-W*LL/2:YL=159: REM ***CENTER UNDER TICK MARK ***
1070  M1=1:M2=1
1080  GOSUB 2500: REM *** DISPLAY A STRING ***
1090  RETURN
1100  REM *** DISPLAY A CAPTION ***
1110  ST$=CA$
1120  IF BD=2 THEN 1210: REM *** BRANCH IF BARS ARE HORIZONTAL ***
1130  REM *** BARS ARE VERTICAL-FORM COORINATES OF FIRST LETTER ***
1140  XL=X1+INT(BW/2):YL=Y0-SC*VA-3
1150  MC=INT(YL/W): REM *** ALLOWED LENGTH ***
1160  IF MC<LEN(ST$) THEN ST$=LEFT$(ST$,MC)
1170  M1=4:M2=2: REM *** SIDEWAYS LETTERING MODE ***
1180  GOSUB 2500: REM *** DISPLAY THE STRING ***
1190  RETURN
1200  REM *** BARS ARE HORIZONTAL - FORM COORDS OF FIRST LETTER ***
1210  XL=X0+SC*VA+3:YL=Y1-INT(BW/2)
1220  MC=INT((279-XL)/W): REM *** ALLOWED LENGTH ***
1230  IF MC<LEN(ST$) THEN ST$=LEFT$(ST$,MC)
1240  M1=1:M2=1: REM *** HORIZONTAL LETTERING MODE ***
1250  GOSUB 2500: REM *** DISPLAY THE STRING ***
1260  RETURN
2500  REM *** DISPLAY A STRING ***
2510  IF LEN(ST$)<1 THEN  RETURN : REM *** NULL STRING ***
2520  FOR I=1 TO LEN(ST$): REM *** CYCLE FOR SUCCESSIVE CHARS. ***
2530  CN=ASC(MID$(ST$,I,1))-31: REM *** INDEX OF CHAR IN FONT DIRECTORY*
2540  S1=DC%(CN-1)+1:S2=DC%(CN): REM *** SEGMENT POINTERS ***
2550  P2=DS%(S1-1): REM ***DISPLAY A CHARACTER ***
2560  NS=S2-S1+1: GOSUB 3200
2570  NEXT I: REM *** END OF CYCLE FOR SUCCESSIVE CHARACTERS ***
2580  RETURN
3000  REM *** LOAD HI-RES CHARACTERS ***
3010  SW%=0
3020  W=6:H=8:A4$=CHR$(4):F1$="LABELTEXT"
3030  HOME : PRINT "FONT FILE(DEFAULT IS ";F1$;"): ": INPUT "";FL$
3040  IF LEN(FL$)=0 THEN FL$=F1$
3050  PRINT A4$;"OPEN ";FL$: PRINT A4$;"READ ";FL$
3060  INPUT "";NC: REM *** NUMBER OF CHARACTERS ***
3070  VTAB 12: FLASH : PRINT "READING IN DATA FROM ";FL$: NORMAL
3080  REM *** CYCLE THROUGH THE CHARACTERS ***
3090  S2=0:P2=0
3100  FOR CN=1 TO NC
3110  S1=S2+1
3120  INPUT "";NS: REM *** NUMBER OF SEGMENTS ***
3130  S2=S1+NS-1:DC%(CN)=S2
3140  GOSUB 3200: REM *** STORE ONE CHARACTER ***
3150  NEXT CN
3160  PRINT A4$;"CLOSE ";FL$: REM *** THE FONT HAS BEEN STORED ***
3170  SW%=1: REM *** SWITCH TO MAKE NEXT SUBROUTINE DISPLAY CHARS. ***
3180  RETURN
```

```
3190  REM *** STORE OR DISPLAY ONE CHARACTER ***
3200  IF NS<1 THEN 4700: REM *** NULL CHARACTER ***
3210  REM *** CYCLE THROUGH THE SEGMENTS ***
3220  FOR SN=S1 TO S2
3230  P1=P2+1: REM *** POINTS TO COORDINATE ARRAY ***
3240  IF SW%=1 THEN 3290: REM *** BRANCH TO DISLAY A CHARACTER
3250  REM *** STORE A CHARACTER ***
3260  INPUT "";NP: REM *** NUMBER OF POINTS ***
3270  P2=P1+NP-1:DS%(SN)=P2: REM *** STARTING POINTER ***
3280  GOTO 4210
3290  P2=DS%(SN):NP=P2-P1+1: REM ***HIGH POINTER TO DISPLAY
3300  REM *** CYCLE THROUGH THE POINTS ***
4210  FOR PN=P1 TO P2
4220  IF SW%<>0 THEN 4250: REM *** BRANCH TO DRAW A CHARACTER ***
4240  INPUT "";PT%(PN,1),PT%(PN,2): GOTO 4610: REM *** BRANCH TO STORE ONE ***
4250  REM *** DISPLAY A SEGMENT ***
4260  IF M2=2 THEN 4300: REM ***  BRANCH FOR SIDEWAYS CHARACTER ***
4270  IF NP=1 THEN  HPLOT XL+PT%(PN,1),YL+PT%(PN,2)
4280  IF PN>P1 THEN HPLOT XL+PT%(PN-1,1),YL+PT%(PN-1,2)TO XL+PT%(PN,1),YL+PT%(PN,2)
4290  GOTO 4610
4300  IF NP=1 THEN  HPLOT XL+H+PT%(PN,2),YL-PT%(PN,1)
4310  IF PN>P1 THEN  HPLOT XL+H+PT%(PN-1,2),YL-PT%(PN-1,1) TO XL+H+PT%(PN,2),YL-PT%
      (PN,1)
4610  NEXT PN,SN
4620  IF SW%=0 THEN  RETURN : REM *** CHAR. HAS BEEN STORED ***
4700  REM *** PROVIDE THE ESCAPEMENT FOR A DISPLACED CHARACTER ***
4710  IF M1=4 THEN 4810: REM *** BRANCH FOR SIDEWAYS WRITING ***
4720  XL=XL+W: RETURN : REM *** INCREASE X FOR NEXT CHARACTER ***
4810  YL=YL-W: RETURN : REM *** REDUCE Y FOR NEXT CHARACTER ***
5000  END
```

5.5 A SLICE OF APPLE PIE

For business presentations to management and clients, pie charts rank with bar charts and line graphs in importance. The PIE CHART program shows the basic principles very simply.

It first displays the main menu, with default values for some of the options:

```
            *** MAIN MENU ***
        OPTION                VALUE(S)
    1. POSITION OF CENTER 60,60
    2. RADIUS OF CHART    50
    3. TITLE OF CHART     SAMPLE PIE CHART
    4. POSITION OF TITLE  20,150
    5. POSITION OF KEY    120,20
    6. INPUT DATA
    7. QUIT

        PLEASE TYPE OPTION:
```

If the defaults are suitable for your chart, just type 6. The menu disappears, a circle of radius 50 appears on the screen, and the program prompts you to type 1 if your data is in percentages, and 2 if it is in actual amounts. If you type 1, it then prompts for the number of slices to be made of the pie. It then prompts for the percentage and the caption of each slice in turn. Lines are drawn from the center of the pie at the correct intervals, proceeding counterclockwise from a horizontal line pointing to the right. The numbers 1, 2, 3, ... are put on the successive slices, and a key is displayed at the left side of the graphics area, to relate the captions and percentages to these numbers (Figure 5.11).

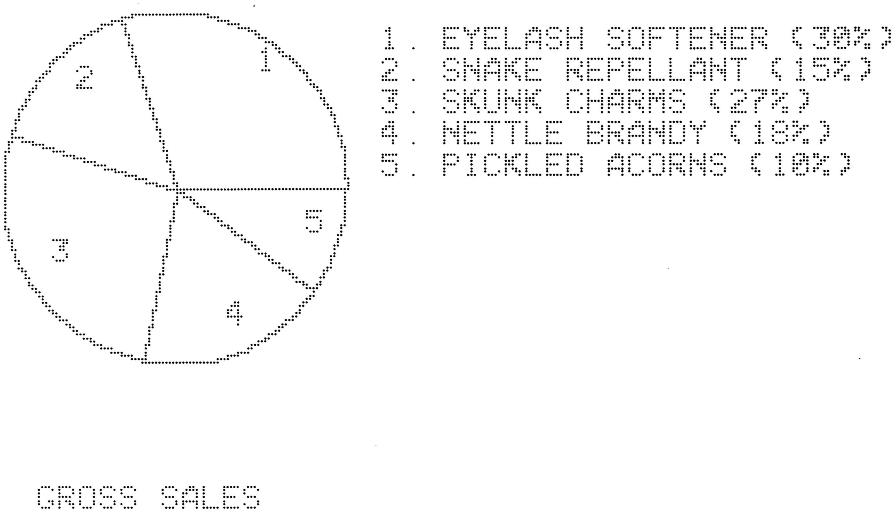


Figure 5.11

If the percentages total more than 100, an error message is displayed. If the total is less than 100, the residual sector is given a numerical label that is identified as OTHER in the key. The chart stays on the screen until you press RETURN. The main menu then reappears.

If you want to specify the actual amounts instead of percentages, type 6 and then 2. The program prompts for the number of slices, for the amount that each will represent, and the caption to be displayed in the key. The chart is drawn, and the amounts and percentages are included in the key, along with the caption (see Figure 5.12).

The main variables in the program are as follows

`PT%(300,2)`	The coordinates that define the font data. These are explained in Section 5.2
`DS%(200)`	The directory of the PT% array (see Section 5.2)
`DC%(70)`	The directory of the DS% array (see Section 5.2)
`H,W`	The height and width of the characters in this font
`XC,YC`	The coordinates of the center of the chart
`R`	The radius of the chart

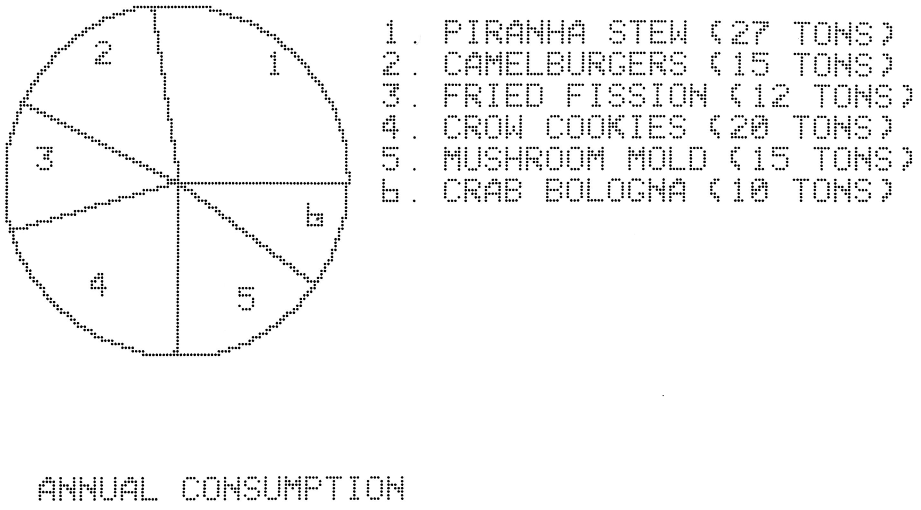


Figure 5.12

`TI$`	The title of the chart
`XT,YT`	The screen coordinates of where the title begins
`XK,YK`	The screen coordinates of the beginning of the key
`F1,F2`	The constants 2π and $\pi/50$
`OP`	The option selected from the main menu
`A$`	The dummy variable in the statement that waits for a key to be pressed
`XL,YL`	The coordinates of the end of a radial line of the chart
`XO,YO`	The coordinates of the start of a short line that approximates an arc of the circumference of the chart
`XN,YN`	The coordinates of the end of this line
`TH`	The polar angle of a point used to draw the circumference or to end a slice of the chart
`MC`	The maximum number of characters for which there is space for an entry in the key
`XL,YL`	The coordinates of the starting point of a string that is drawn on the screen
`ST$`	The actual string
`TP`	The growing sum of the percentages of items that have been input
`TP`	The polar angle of the starting line of the current segment
`OZ`	This is set to 1 and 2, respectively, when the user wishes to type percentages or actual quantities
`NI`	The number of items for which percentages or amounts are typed
`IN`	The index of the loops that deal with the successive items
`PE`	The percentage for an item
`CA$`	The word(s) to identify an item in the key

`MP`	The polar angle of the median line of the current segment
`UN$`	The units (for example, tons) in which the amounts are measured, when the user inputs amounts instead of percentages
`SU`	The growing sum of the amounts for the successive items
`AM(IN)`	The INth amount

The program can be followed by reference to the list of variables and the REM statements. Each entry in the key is formed by concatenating (1) the reference number, (2) a period, (3) the caption (which is truncated if necessary), and (4) the percentage or the amount, in parentheses. The unit of measure is included when amounts are specified.

The parts of PIE CHART that read font data and display strings of characters are based on parts of FONT FORMER, like the corresponding portions of BAR CHART. These parts of the programs use variables that are explained in Section 5.2. The REM statements in BAR CHART provide a further explanation of the font operations. The same line numbers are used for the parts that read font data and display strings in FONT FORMER, BAR CHART, and PIE CHART.

Suggested variations

1. Let the program complete a chart when the E key is pressed, without requiring the number of slices to be typed at the start.
2. Allow a third method of specifying data in which the total is given, then the amounts to be represented by one or more slices, with the balance to be a slice labeled OTHER if it is not reached by the final item.
3. Let the user specify the angle that the "more clockwise" line of the first slice makes with the horizontal (for example, –15 in Figure 5.13).

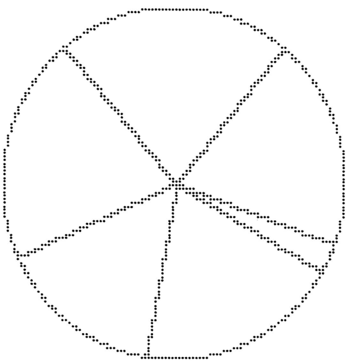


Figure 5.13

4. Allow captions to be displayed horizontally either (a) inside the slices, (b) just outside the slices, or (c) in a key, as a choice of the user.
5. Allow one or more slices to be moved out slightly (Figure 5.14).

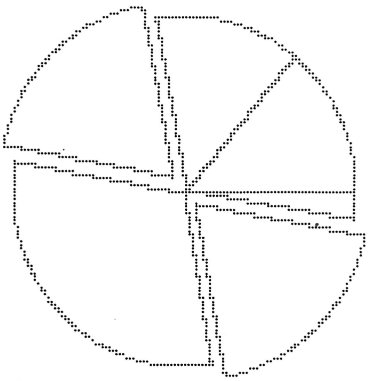


Figure 5.14

6. Give a three-dimensional effect, as in Figure 5.15.

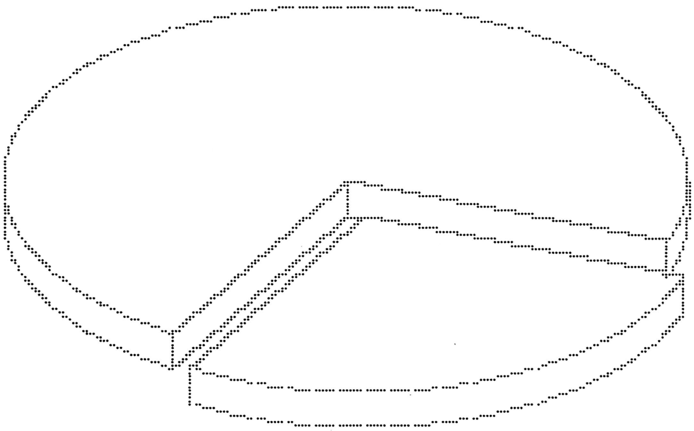
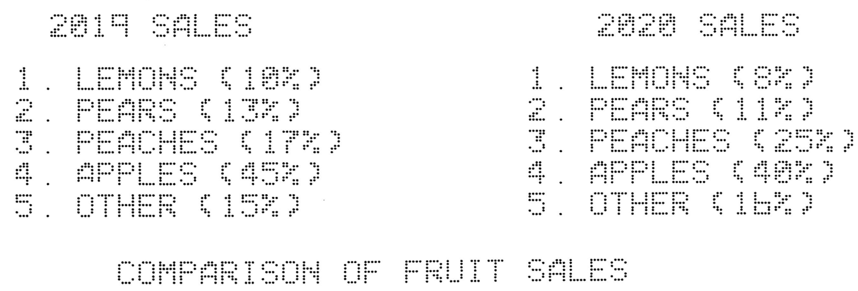


Figure 5.15

7. Allow several pie charts to be displayed at the same time (Figure 5.16).



```
   2019 SALES                      2020 SALES

1. LEMONS (10%)               1. LEMONS (8%)
2. PEARS (13%)                2. PEARS (11%)
3. PEACHES (17%)              3. PEACHES (25%)
4. APPLES (45%)               4. APPLES (40%)
5. OTHER (15%)                5. OTHER (16%)

      COMPARISON OF FRUIT SALES
```

Figure 5.16

The PIE CHART program

```
 100  REM *** PIE CHART ***
 110  LOMEM: 16384: REM *** PROTECT MEMORY ***
 120  DIM DC%(70),DS%(200),PT%(360,2):H=8:W=6: REM *** FOR HI-RES FONT ***
 130  GOSUB 3000: REM *** LOAD HI-RES FONT ***
 140  XC=60:YC=60:R=50: REM *** DEFAULT CENTER AND RADIUS OF CHART ***
 150  TI$="SAMPLE PIE CHART":XT=20:YT=150: REM *** DEFAULT TITLE AND POSITION ***
 160  XK=120:YK=20: REM *** DEFAULT POSITION OF KEY ***
 170  F1=6.283:F2=.06283: REM *** 2 PI AND 2 PI/100 ***
 180  TEXT : HOME
 190  PRINT  TAB( 10);"*** MAIN MENU ***"
 200  PRINT  TAB( 5);"OPTION"; SPC( 15);"VALUE(S)"
 210  PRINT " "
 220  PRINT "1. POSITION OF CENTER"; TAB( 23);XC;", ";YC
 230  PRINT "2. RADIUS OF CHART"; TAB( 23);R
 240  PRINT "3. TITLE OF CHART"; TAB( 23);TI$
 250  PRINT "4. POSITION OF TITLE"; TAB( 23);XT;", ";YT
 260  PRINT "5. POSITION OF KEY"; TAB( 23);XK;", ";YK
 270  PRINT "6. INPUT DATA"
 280  PRINT "7. QUIT"
 290  VTAB 12: HTAB 5: CALL -958: REM *** CLEAR REST OF SCREEN ***
 300  INPUT "PLEASE TYPE OPTION : ";OP
 310  ON OP GOTO 340,350,360,370,380,390,1000
 320  PRINT "INVALID OPTION :";OP
 330  PRINT "PRESS ANY KEY TO CONTINUE": GET A$: GOTO 290
 340  INPUT "COORDINATES OF CENTER: ";XC,YC: GOTO 180
 350  INPUT "RADIUS OF CHART: ";R: GOTO 180
 360  INPUT "TITLE: ";TI$: GOTO 180
 370  INPUT "POSITION OF TITLE: ";XT,YT: GOTO 180
 380  INPUT "POSITION OF KEY: ";XK,YK: GOTO 180
 390  HGR : HCOLOR=3: REM *** PREPARE TO PLOT ***
 400  IF XC>R AND YC>R THEN 420: REM *** BRANCH IF CHART WILL FIT ***
 410  VTAB =21: PRINT "CHART WOULD NOT FIT ON THE SCREEN": GOTO 330
 420  XO=XC+R:YO=YC: REM *** STARTING POINT ON THE CIRCLE ***
 430  FOR TH=0 TO F1 STEP F2: REM *** CYCLE AROUND THE CIRCLE ***
 440  XN=XC+R*COS(TH):YN=YC-R*SIN(TH)
 450  HPLOT XO,YO TO XN,YN: REM *** DRAW THE APPROXIMATION TO THE ARC ***
 460  XO=XN:YO=YN
 470  NEXT TH
 480  MC=INT((280-XK)/W-8): REM *** LIMIT THE LENGTH OF ENTRIES IN THE KEY ***
 490  HPLOT XC,YC TO XC+R,YC: REM *** DRAW THE FIRST RADIAL LINE ***
 500  PT=0: REM *** INITIALIZE THE ANGLE OF THE STARTING LINE OF THE SLICE ***
 510  XL=XT:YL=YT:ST$=TI$: GOSUB 2500: REM *** DRAW THE TITLE ***
 520  TP=0: REM *** INITIALIZE THE CUMULATIVE PERCENTAGE ***
 530  VTAB 21: CALL -958: PRINT "TYPE 1 TO INPUT PERCENTAGES,"
 540  INPUT "      2 TO INPUT AMOUNTS: ";OZ
 550  IF OZ<>1 THEN 780: REM *** BRANCH TO TYPE AMOUNTS ***
 560  VTAB 21: CALL -958: INPUT "NUMBER OF ITEMS: ";NI
 570  FOR IN=1 TO NI: REM *** CYCLE FOR SUCCESSIVE SLICES ***
 580  VTAB 22: CALL -958: INPUT "PERCENTAGE AND CAPTION: ";PT,CA$
 590  TP=TP+PT: REM *** INCREASE THE CUMULATIVE PERCENTAGE ***
 600  IF TP<=100 THEN 640: REM *** BRANCH IF VALID ***
 610  VTAB 23: CALL -958: PRINT "TOTAL EXCEEDS 100 PERCENT"
 620  PRINT "PRESS ANY KEY TO RETYPE LAST ENTRY"
 630  TP=TP-PT: GET A$: GOTO 580: REM *** TRY AGAIN ***
 640  TH=TP*F2: REM *** ANGLE IN RADIANS OF OUTER EDGE OF CURRENT SLICE ***
 650  HPLOT XC,YC TO XC+R*COS(TH),YC-R*SIN(TH): REM *** DRAW IT ***
 660  MP=TH-PT*F2/2: REM *** ANGLE OF MEDIAN LINE OF SLICE ***
 670  XL=XC+(R-9)*COS(MP):YL=YC-(R-9)*SIN(MP): REM *** COORDS OF REF NUMBER ***
 680  ST$=STR$(IN): GOSUB 2500: REM *** DRAW THE REFERENCE NUMBER ***
 690  XL=XK:YL=YK+(IN-1)*(H+1): REM *** COORDINATES OF KEY ***
 700  ST$=ST$+". "+LEFT$(CA$,MC)+" ("+STR$(PT)+"%)": REM *** FORM THE ENTRY ***
 710  GOSUB 2500: REM *** DRAW IT ***
 720  PT=TP: REM *** RESET ANGLE OF STARTING LINE FOR NEXT SEGMENT ***
 730  NEXT IN:: REM *** END OF CYCLE FOR SUCCESSIVE SLICES ***
 740  VTAB 22: CALL -958: REM *** MOVE CURSOR AND CLEAR THE SCREEN ***
 750  PRINT "PRESS ANY KEY TO RETURN TO MAIN MENU": GET A$
 760  GOTO 180
 770  REM *** THE USER INPUTS THE AMOUNTS ***
 780  INPUT "UNITS: ";UN$:MC=MC-LEN(UN$)-2: REM *** ALLOW FOR UNITS IN THE KEY ***
 790  INPUT "NUMBER OF ITEMS: ";NI
 800  SU=0:PT=0:REM *** INITIALIZE THE TOTAL ***
 810  FOR IN=1 TO NI: REM *** CYCLE FOR SUCCESSIVE AMOUNTS ***
 820  VTAB 23: CALL -958: INPUT "AMOUNT AND CAPTION: ";AM(IN),CA$(IN)
 830  SU=SU+AM(IN): REM *** INCREASE THE TOTAL&FORM THE KEY ***
 840  CA$(IN)=STR$(IN)+". "+LEFT$(CA$(IN),MC)+" ("+STR$(AM(IN))+" "+UN$+")"
 850  NEXT IN: REM *** END CYCLE FOR SUCCESSIVE AMOUNTS ***
 860  FOR IN=1 TO NI: REM *** CYCLE FOR SUCCESSIVE SLICES ***
 870  TP=TP+(AM(IN)/SU*100): REM *** INCREASE THE CUMULATIVE PERCENTAGE ***
 880  TH=TP*F2: REM *** CONVERT TO RADIANS ***
 890  HPLOT XC,YC TO XC+R*COS(TH),YC-R*SIN(TH): REM *** DRAW THE EDGE ***
 900  MP=(TH+PT)/2: REM *** ANGLE OF MEDIAN LINE OF SLICE ***
 910  XL=XC+(R-9)*COS(MP): REM *** COORDINATES FOR ENTRY IN KEY ***
 920  YL=YC-(R-9)*SIN(MP)
 930  PT=TH: REM *** FIRST EDGE FOR NEXT SLICE ***
 940  ST$=STR$(IN): GOSUB 2500: REM *** DRAW THE SLICE NUMBER ***
 950  XL=XK:YL=YK+(IN-1)*(H+1): REM *** COORDS OF ENTRY IN KEY ***
 960  ST$=CA$(IN): GOSUB 2500: REM *** DRAW THE ENTRY ***
 970  NEXT IN: REM *** END OF CYCLE FOR SUCCESSIVE SLICES ***
 980  PRINT "PRESS ANY KEY TO CONTINUE";: GET A$: GOTO 180
1000  END : REM *** QUIT ***
2500  REM *** DISPLAY A STRING ***
2510  IF LEN(ST$)<1 THEN  RETURN
2520  FOR I=1 TO LEN(ST$)
2530  CN=ASC(MID$(ST$,I,1))-31
2540  S1=DC%(CN-1)+1:S2=DC%(CN)
2550  P2=DS%(S1-1)
```

```
2560  NS=S2-S1+1: GOSUB 3200
2570  NEXT I
2580  RETURN
3000  REM *** LOAD AND DISPLAY CHARACTERS ***
3010  SW%=0
3020  W=6:H=8:A4$=CHR$(4):F1$="LABELTEXT"
3030  HOME : PRINT "FONT FILE (DEFAULT IS ";F1$;"): ";: INPUT "";FL$
3040  IF LEN(FL$)=0 THEN FL$=F1$
3050  PRINT A4$;"OPEN ";FL$: PRINT A4$;"READ ";FL$
3060  INPUT "";NC
3070  VTAB 12: FLASH : PRINT "READING DATA FROM ";FL$: NORMAL
3080  REM *** CYCLE THROUGH THE CHARACTERS ***
3090  S2=0:P2=0
3100  FOR CN=1 TO NC
3110  S1=S2+1
3120  INPUT "";NS
3130  S2=S1+NS-1:DC%(CN)=S2
3140  GOSUB 3200
3150  NEXT CN
3160  PRINT A4$;"CLOSE ";FL$
3170  SW%=1
3180  RETURN
3190  REM *** STORE OR DISPLAY ONE CHARACTER ***
3200  IF NS<1 THEN 4700
3210  REM : CYCLE THROUGH THE SEGMENTS
3220  FOR SN=S1 TO S2
3230  P1=P2+1
3240  IF SW%=1 THEN 3290
3250  REM *** STORE A CHARACTER ***
3260  INPUT "";NP
3270  P2=P1+NP-1:DS%(SN)=P2
3280  GOTO 4210
3290  P2=DS%(SN):NP=P2-P1+1
3300  REM *** CYCLE THROUGH THE POINTS ***
4210  FOR PN=P1 TO P2
4220  IF SW%<>0 THEN 4250
4240  INPUT "";PT%(PN,1),PT%(PN,2): GOTO 4610
4250  REM *** DISPLAY A SEGMENT ***
4270  IF NP=1 THEN  HPLOT XL+PT%(PN,1),YL+PT%(PN,2)
4280  IF PN>P1 THEN HPLOT XL+PT%(PN-1,1),YL+PT%(PN-1,2)TO XL+PT%(PN,1),YL+PT%(PN,2)
4610  NEXT PN,SN
4620  IF SW%=0 THEN  RETURN
4700  REM *** PROVIDE THE ESCAPEMENT ***
4720  XL=XL+W: RETURN
5000  REM ***  END  ***
```

Chapter 6

A medley of motley curves

This chapter moves on from applications that just use straight lines and circles to problems that involve curves of many other kinds. Footballs and other projectiles follow parabolic paths, which are simple examples of the polynomial functions that pervade most areas in which mathematics is applied to daily life. Some related curves are the basis of a process called Bezier fitting that engineers use interactively in automobile design. Satellites, both natural and manmade, follow elliptical orbits. Other curves, called cycloids, were used to explain celestial behavior before modern theories evolved. Sine waves are at the heart of electrical and many other phenomena. Programs that produce and use all these kinds of curves are described in this chapter.

6.1 ON THE PATH OF A FLYING SAUCER

If you need practice dodging saucepans or catching baseballs or writing programs for realistic games, the program GALILEO may give you some help. It displays the movement of a projectile that is thrown or fired at an angle from the ground. The program prompts for (1) this angle, (2) the size of the projectile (that is, the number of pixels in each side of a small square that depicts it), and (3) the starting velocity. The program uses units that make the trajectory flatten out at the top of the screen, and reach the ground at the right edge, when the angle of elevation is 66.5° and the initial velocity v is 110 feet per second. A small square appears and disappears at a succession of points along the appropriate path, which is in fact a parabola. Galileo demonstrated this nearly 400 years ago. Figures 6.1A–6.1C are typical trajectories.

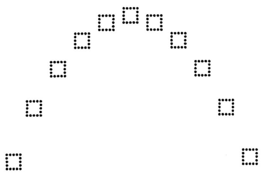


Figure 6.1A. Default.

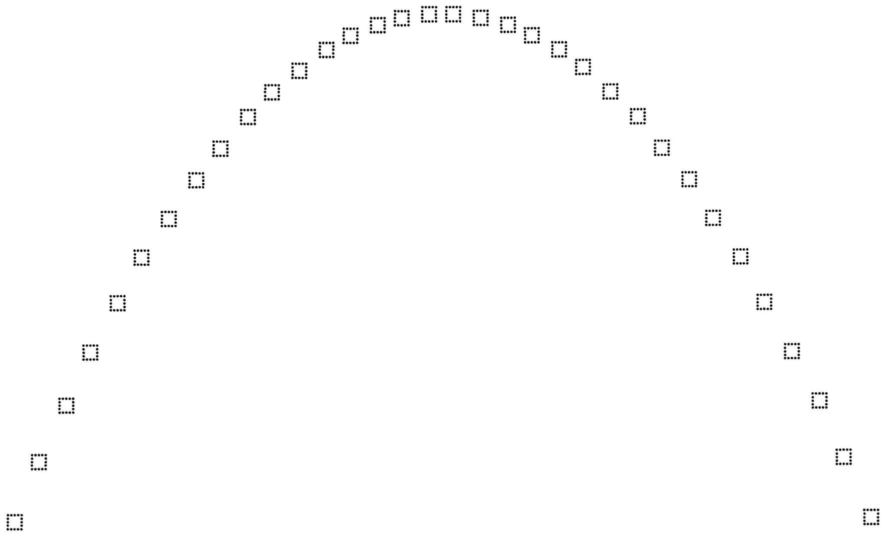


Figure 6.1B. v=105.

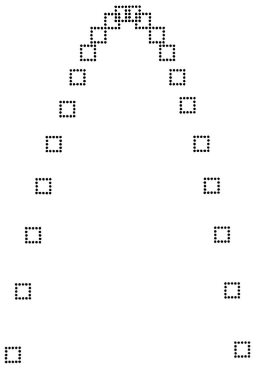


Figure 6.1C. v=80, angle=80°.

After plotting a trajectory, the program clears the screen and repeats the initial prompt. See what happens when you type different speeds and angles in response. The program does not take wind resistance into account, so the range and maximum height are both proportional to the square of v. Thus, if you make v=55.5 feet per second, keeping the initial angle at 66.5°, the projectile rises to just a quarter of the height of the screen, and ends its flight a quarter of the way across.

The effect of changing the angle depends on a little trigonometry. If $v=100$, the ranges and heights for different angles are as follows:

angle:	0	10	20	30	40	50	60	70	80	90
range:	0	106	200	269	306	306	269	200	106	0
height:	0	5	18	39	64	91	116	137	150	155

If you want to know how they were found, the formulas are very simple. If a is the initial angle, g is the gravitational constant, and t is the time, then

range $R=v^2 \sin 2a/g,$

maximum height $H=(v \sin a)^2/2g$

x coordinate after t seconds: $vt \cos a$

y coordinate after t seconds: $vt \sin a-gt^2/2$

y coordinate in terms of x: $y=(4H/R^2)x(x-R)$

The main variables of the program are

`A`	The angle of elevation
`V`	The initial velocity
`S`	The size of the projectile
`G`	The gravitational constant g
`HG`	Half of g
`CF`	The factor to convert from degrees to radians
`AR`	The angle of elevation measured in radians
`OP`	The option selected from the menu
`HC,VC`	The horizontal and vertical components of initial velocity
`TF`	The time of flight
`HS`	Half the side length of the projectile in screen units
`DT`	The time interval for the simulation
`T`	The time that the projectile has been in flight
`CO`	The color—that is, 3 to draw and 0 to erase
`K`	The index of the loops that control the simulation speed

After displaying the initial prompt, the program goes through a cycle that computes the successive positions of the projectile at equal intervals of time, and displays it accordingly, erasing it from its previous position as well. The x and y values are computed from the formulas that have just been given.

Suggested variations

1. Draw a small cannon at the left of the screen that points in the required direction and fires the projectile.
2. Make the projectile go across the screen from right to left.
3. Fire the projectile at an object that stays still, and disappears with a grunt or a groan when it is hit (see ROBOT in Section 2.4).
4. Make it disappear in pieces.
5. Make it move about while it is being shot at.
6. Let the user move the gun as well.
7. Have guns firing at each other across the screen.
8. Make the projectile a harmless ball, thrown over a net.
9. Make the net move.

10. Make the ball bounce, with the same initial angle and speed as when it started.
11. Reduce the speed at each bounce by a percentage the user specifies.
12. Use a shape definition for the projectile.
13. Fire a succession of projectiles across the screen, at angles that go up and down.
14. If your Apple has paddles, use them to control the speed and elevation of the projectiles.

The GALILEO program

```
100  REM *** GALILEO-PLOTS A TRAJECTORY ***
110  A=66.5:V=55.5:S=4: REM *** DEFAULT LAUNCH ANGLE, VELOCITY AND SIZE ***
120  G=32.2:HG=G/2:CF=57.2958: REM *** PHYSICAL CONSTANTS ***
130  POKE 34,23: REM *** PROTECT THE MENU FROM SCROLLING ***
140  TEXT : HOME : HGR: HCOLOR=3
150  REM *** MAIN MENU ***
160  VTAB 21: CALL -958: REM *** CLEAR THE TEXT AREA ***
170  PRINT "*** OPTIONS AND CURRENT VALUES ***"
180  PRINT "1. ANGLE (";A;"), 2. VELOCITY (";V;")"
190  PRINT "3. SIZE (";S;"), 4. DISPLAY PATH, 5. QUIT"
200  REM *** PROMPT FOR INPUT ***
210  VTAB 24: CALL -958: INPUT "TYPE AN OPTION? ";OP
220  VTAB 24: CALL -958: REM *** ERASE THE LAST LINE ***
230  ON OP GOTO 260,270,280,290,430
240  VTAB 24: PRINT "INVALID OPTION"
250  FOR X=1 TO 1000: GOTO 210
260  INPUT "FIRING ANGLE? ";A: GOTO 160
270  INPUT "VELOCITY? ";V: GOTO 160
280  INPUT "SIZE OF PROJECTILE? ";S: GOTO 160
290  REM *** COMPUTE DATA FOR PROJECTILE'S PATH ***
300  AR=A/CF:HC=V*COS(AR):VC=V*SIN(AR)
310  IF S>8 THEN S=8
320  TF=VC/HG:HS=S/2
330  DT=TF/(2+INT(V*TF/20))-.0001: REM *** ALLOW 20 SITINGS ***
350  FOR T=0 TO TF STEP DT: REM *** SIMULATE THE PASSAGE OF TIME ***
360  X=5+T*HC
370  Y=154-T*(VC-T*HG)
380  CO=3: GOSUB 440: REM ***  DRAW THE PROJECTILE ***
390  FOR K=1 TO 50: NEXT K: REM *** LEAVE IT ON THE SCREEN BRIEFLY ***
400  CO=0: GOSUB 440: REM *** ERASE IT ***
410  NEXT T: REM *** END THE SIMULATION CYCLE ***
420  GOTO 150
430  TEXT : HOME : END
440  HCOLOR=CO: REM *** DRAW OR ERASE THE PROJECTILE ***
450  IF X-HS<0 OR X+HS>279 THEN  POP : GOTO 160: REM *** START AGAIN ***
460  IF Y-HS<0 OR Y-HS>159 THEN  RETURN : REM *** BYPASS THIS VALUE ***
470  HPLOT X-HS,Y-HS TO X-HS,Y+HS TO X+HS,Y+HS TO X+HS,Y-HS TO X-HS,Y-HS
480  FOR K=1 TO 100: NEXT K: REM *** DELAY BEFORE THE NEXT ACTION ***
490  RETURN
```

6.2 CURVES THAT GO UP AND DOWN

You can plot more parabolas, like the curve that the projectile follows, and a host of other curves that go up and down several times, using the POLYPLOT program. Some examples are shown in Figures 6.2A–6.2D. The *y* coordinate of each point on one of these curves is found by adding two or more terms that contain the *x* coordinate squared, cubed, or raised to some other positive power. Functions that are constructed this way are called polynomials.

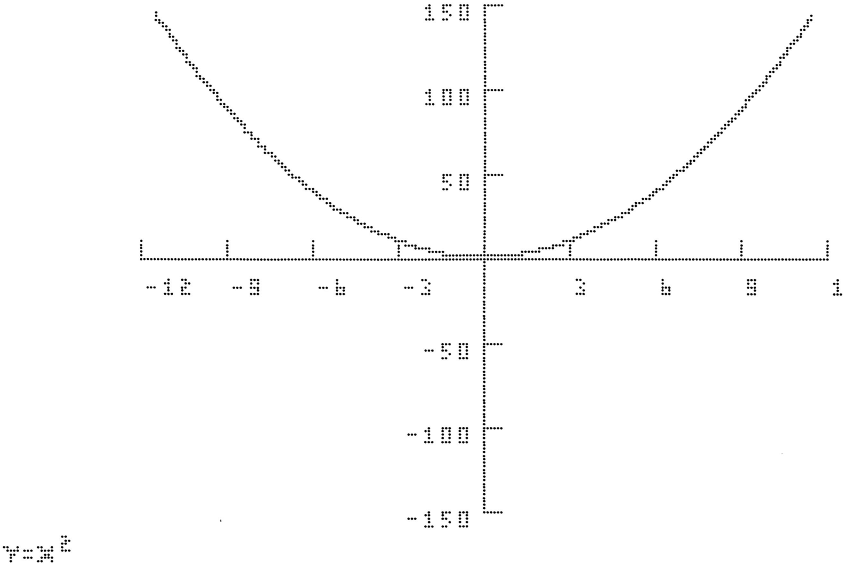


Figure 6.2A

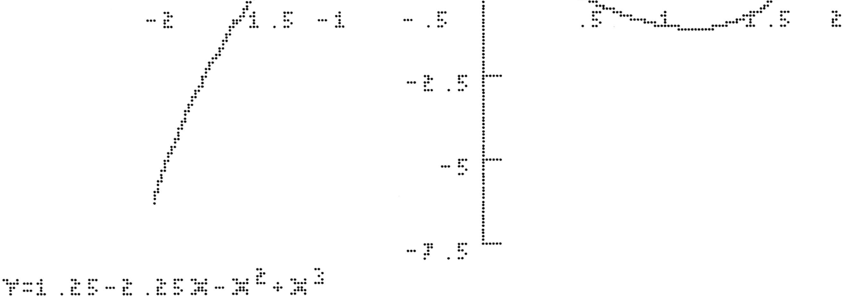


Figure 6.2B

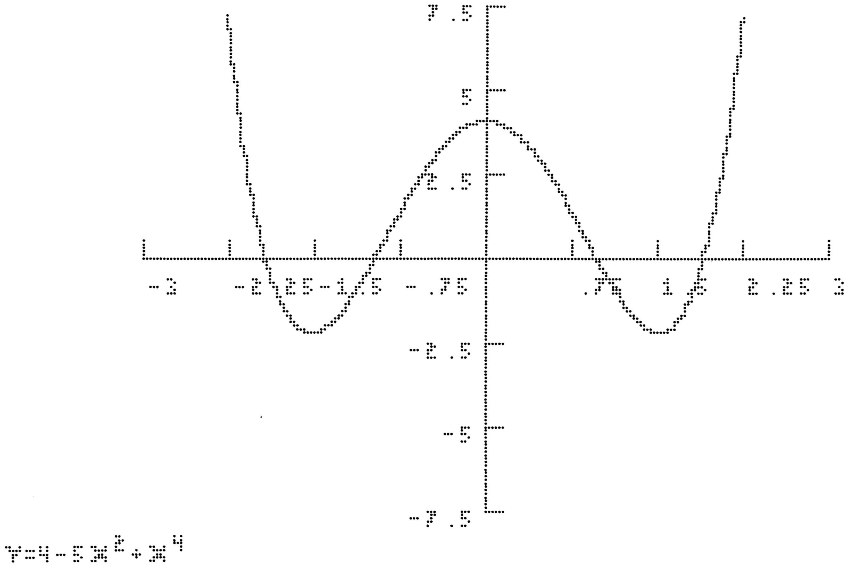


Figure 6.2C

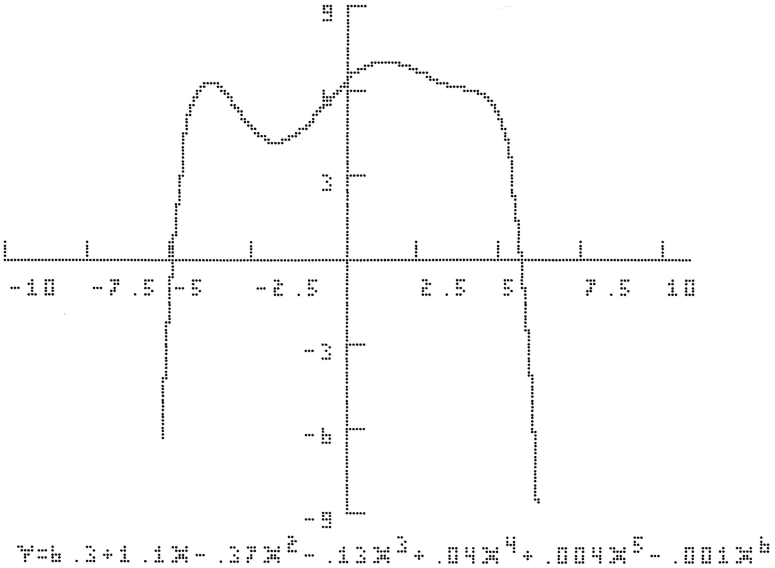


Figure 6.2D

The program begins by displaying the main menu:

```
TYPE 1 FOR SAMPLE POLYNOMIALS
     2 TO SPECIFY A POLYNOMIAL
     3 TO QUIT:
```

If you type 1, the program offers some more options:

```
TYPE 1 FOR QUADRATICS,  2 FOR CUBICS,
     3 FOR OTHERS,  4 FOR MAIN OPTIONS:
```

If you type 1, the program displays the graph of $y=3+2x+0.1x^2$ (Figure 6.3). The x axis goes from -48 to 48, and the y axis from -60 to 60. Labeled tick marks are drawn on the x axis at eight equal intervals, and on the y axis at six equal intervals. The equation for the curve is shown on the screen too.

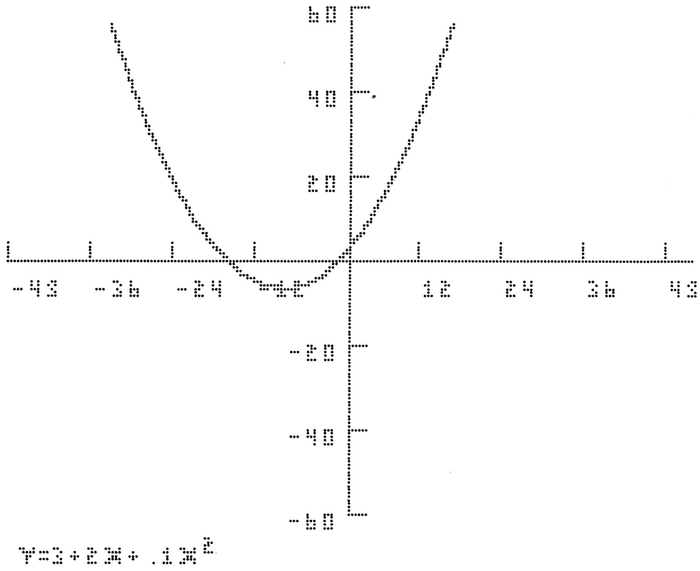


Figure 6.3

The program then prompts

```
TYPE 1 FOR ANOTHER QUADRATIC,
      2 TO RETURN TO MAIN MENU:
```

If you type 1, the program plots a graph of $1-2x-2x^2$ (Figure 6.4). This time the x axis goes from −8 to 8, and the y axis from −30 to 30. Labeled tick marks are drawn at eight and six intervals, respectively, and the equation is displayed as well.

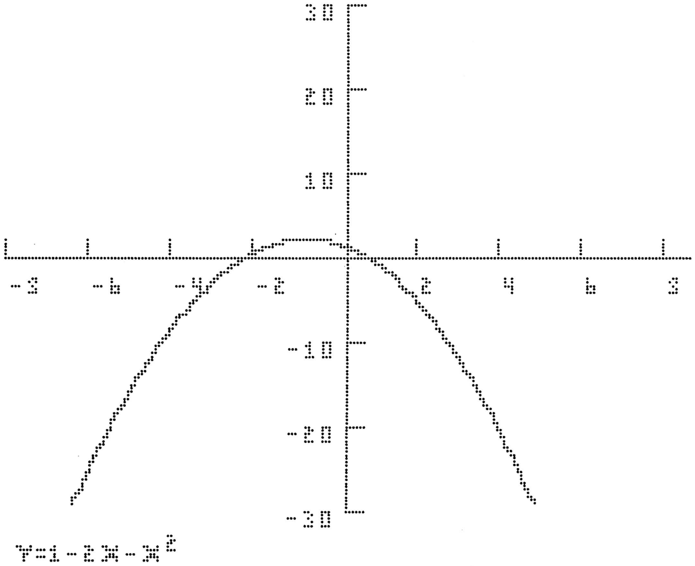


Figure 6.4

If you continue to type 1 after each display has appeared, you get the graphs shown in Figures 6.5A–6.5D in succession, with labeled tick marks on the axes, and the equations at the bottom of the screen. If you type 1 again for another quadratic, the program goes back to the one it drew first, and repeats the sequence if you continue to type 1 after each is drawn.

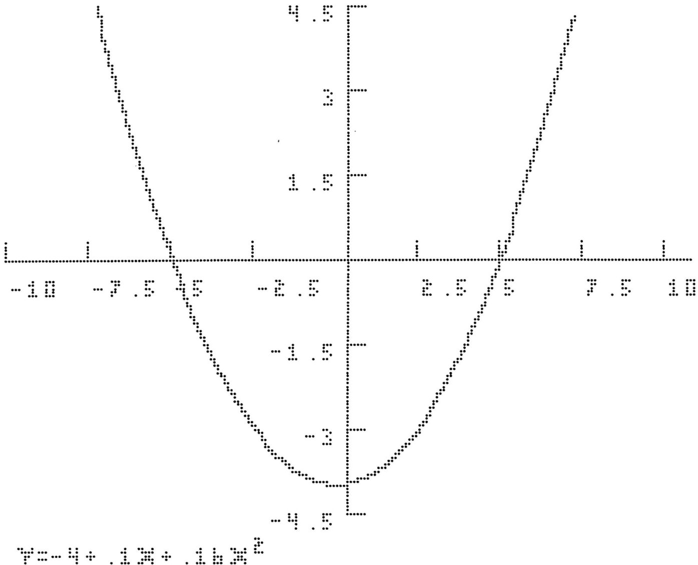


Figure 6.5A

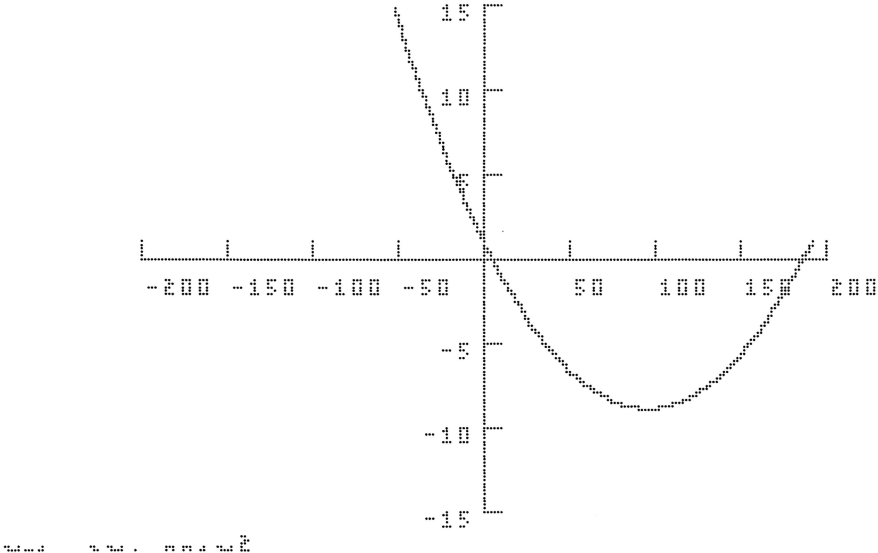


Figure 6.5B

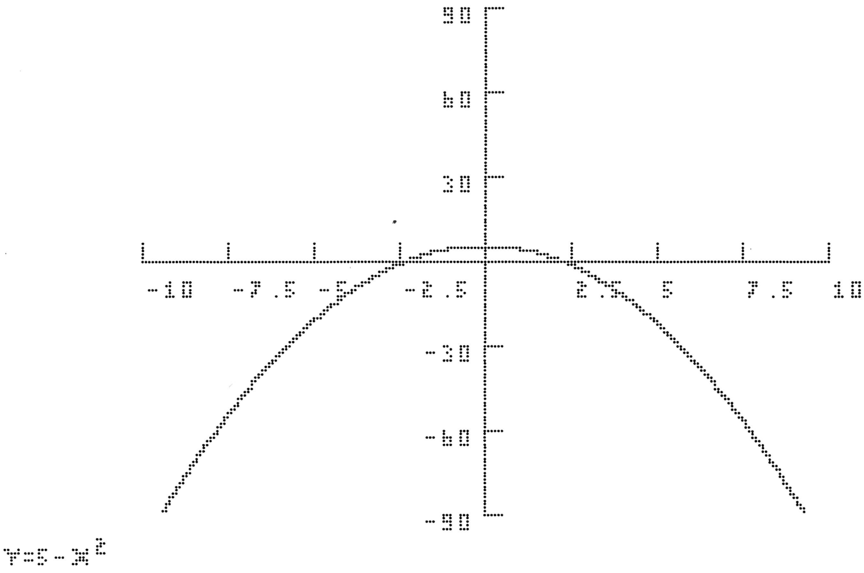


Figure 6.5C

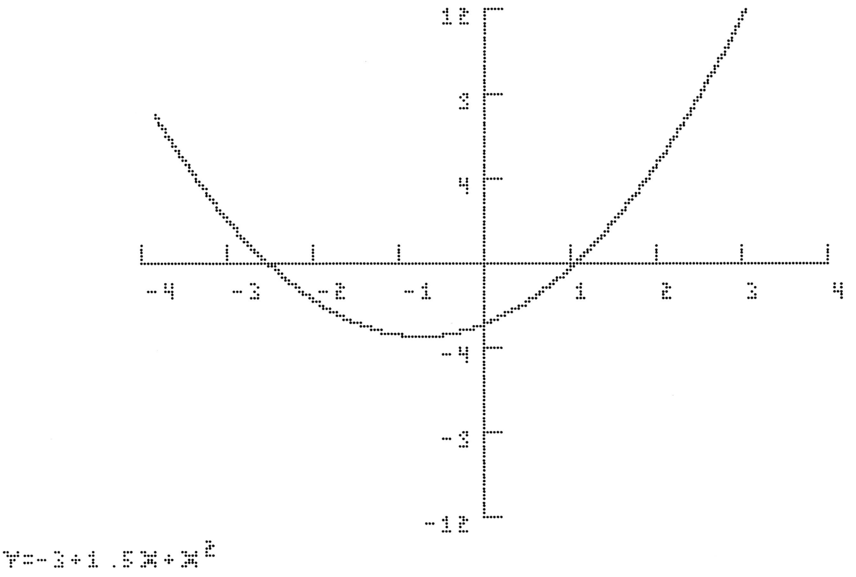


Figure 6.5D

If however you (1) return to the main menu by typing 2, (2) select option 1 again to see some more polynomials, and (3) select option 2, for cubics, in response to the successive prompts, the program displays the cubic curve (Figure 6.6) and the message

```
TYPE 1 FOR ANOTHER CUBIC,
     2 TO RETURN TO MAIN MENU:
```

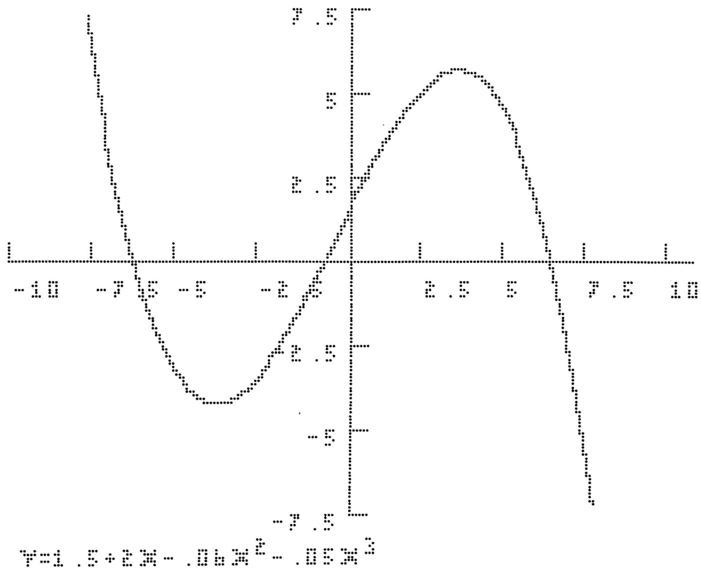


Figure 6.6

You can get a succession of cubic curves by typing 1 each time this prompt reappears (Figures 6.7A–6.7C). These curves all flatten out twice, or exceptionally just once as in the last example. Labeled tick marks are drawn on the axes and the equations are displayed as well. The curves are called cubics because, in the equations for y, the highest power of x is 3.

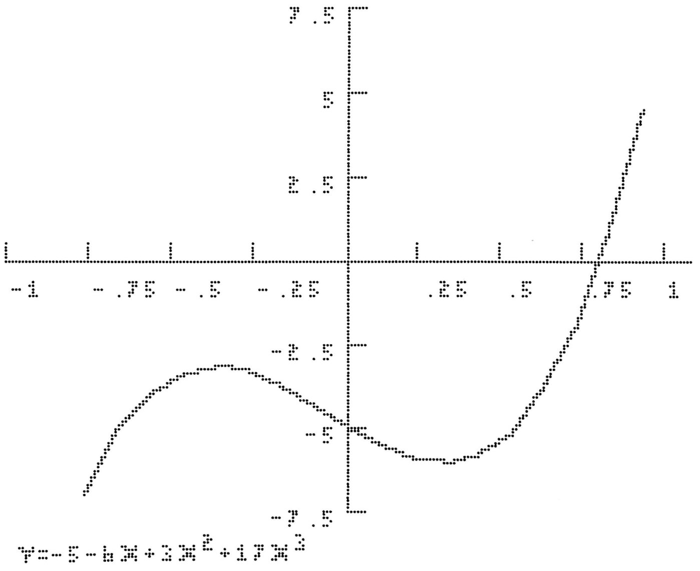


Figure 6.7A

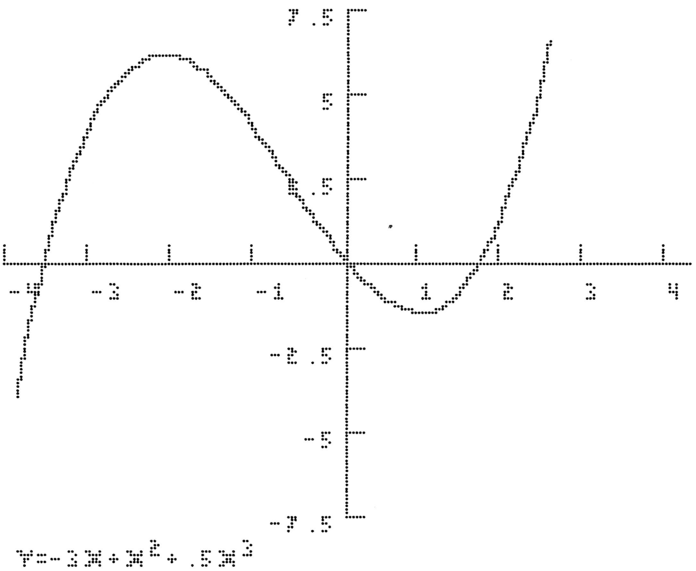


Figure 6.7B

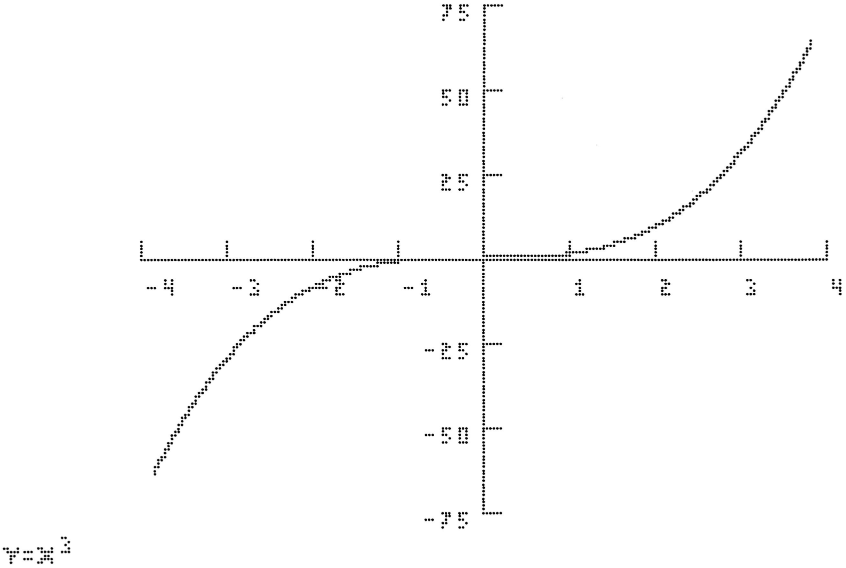


Figure 6.7C

If you (1) return to the main menu by typing 2, (2) select option 1 again to see still more polynomials, and (3) select option 3, for OTHERS, in response to the successive prompts, the program displays the graph of a quintic equation (Figure 6.8) and the message

```
TYPE 1 FOR ANOTHER POLYNOMIAL,
     2 TO RETURN TO MAIN MENU:
```

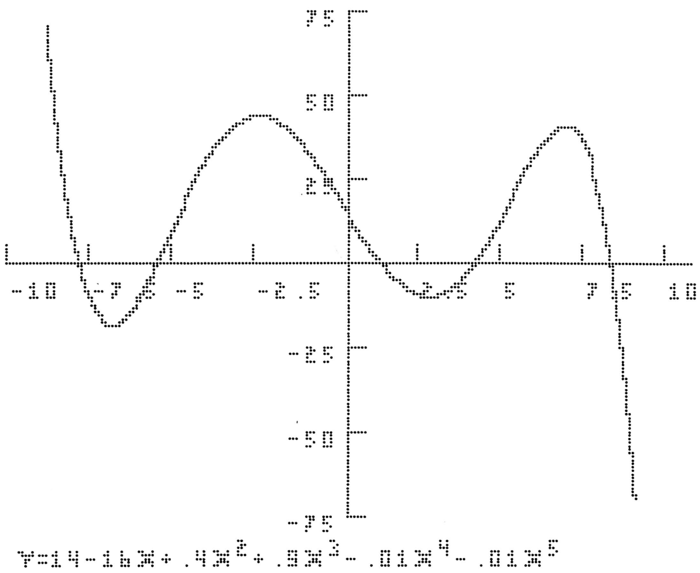


Figure 6.8

If you continue to type 1 after each display appears, you get a succession of graphs that go up and down, on axes with labeled tick marks, and the equation at the bottom of the screen. The prompt for further action appears in the text area too. The highest power of *x* in each equation equals or exceeds the number of times the corresponding graph flattens out.

You may want to define some polynomials of your own. Return to the main menu, and type 2. The program prompts in turn for the degree of the polynomial (that is, the highest power), the constant term, and the further coefficients, the tick mark intervals on the *x* and *y* axes, and the plotting interval in the *x* direction. The *x* axis is drawn between limits that are minus and plus four times the *x* tick mark interval. The *y* axis is drawn between limits that are minus and plus three times the *y* tick mark interval. The program clips any portions where *y* is outside the range of the *y* axis. It flashes a message while plotting is in progress.

After the polynomial has been plotted, the program prompts

```
    TYPE 1 TO SPECIFY ANOTHER POLYNOMIAL,
           2 TO RETURN TO MAIN MENU:
```

If you type 1, it repeats the sequence of actions that have just been described. If you type 2, it returns to the main menu.

The main variables in the program are as follows

`CO(IQ)`	The constant in the IQth preset quadratic
`C1(IQ)`	The coefficient of x in the IQth quadratic
`C2(IQ)`	The coefficient of x^2 in the IQth quadratic
`AX(IQ)`	The tick mark intervals along the x axis for this plot, that is, one quarter of the maximum value of x
`BY(IQ)`	The tick mark intervals along the y axis, that is, one third of the maximum value of y
`D(IQ)`	The x interval for plotting this quadratic

`H0(IQ),H1(IQ),H2(IQ),H3(IQ)`	The constant and coefficients in the IQth preset cubic
`EX(IQ),FY(IQ),G(IQ)`	The tick mark intervals along the *x* and *y* axes for this cubic and the *x* interval for plotting it
`JX(IQ),KY(IQ),GG(IQ)`	The corresponding data for the IQth preset polynomial of higher degree
`CD(IR)`	The IRth coefficient in the polynomial being plotted
`IQ`	The index of the successive preset polynomials of similar kind (that is, quadratics, cubics, higher degree)
`CC(IR)`	The IRth number in the representation of the preset higher degree polynomials. Each polynomial is represented by its degree followed by the constant and the coefficients of all the terms up to the highest. Every constant and/or coefficient that is zero must be included
`IM(1),IM(2),IM(3)`	The numbers of preset quadratics, cubics and higher degree polynomials, respectively
`IR`	The index of the CC array
`DG%`	The degree of the polynomial
`IS`	The index of the successive terms starting at 0 for the constant
`WD$(1),WD$(2),WD$(3)`	The words QUADRATIC, CUBIC and POLYNOMIAL
`W,H`	The width and height of the characters to be drawn
`HH`	Half the height
`OP`	The option that the user selects from the main menu to request preset polynomials or to specify another polynomial
`OQ`	The option that the user selects from the secondary menu to be able to specify the degree of the preset polynomials
`XD,YD`	The tick mark intervals along the *x* and *y* axes for the current plot
`DP`	The *x* interval for the current plot
`TN`	The index of the coefficients in the cycle that prompts for these in a polynomial that the user specifies
`XL,YL`	The screen coordinates where the next character will be appear—XL is increased by the width of the character whenever one is drawn
`SW%`	A switch to supress the + sign at the beginning of the expression for the polynomial
`XB%`	The *x* position on the screen at the beginning of the current term—the program backtracks to this position if it finds later that there is not enough room to complete the term
`NT`	This indexes the successive coefficients starting with the constant in the cycle that draws the equation on the screen
`ST%`	A character string to be drawn

`I`	The index of successive characters in the string that is being drawn
`EX,MA`	The exponent and mantissa in a floating point number
`II`	The index of the cycle that provides the appropriate number of zeros before the significant digits when a floating point number with a negative exponent is converted to fixed point form
`SX,SY`	The scaling factors to convert from user to screen coordinates
`XU,YU`	The x and y values of a point to be plotted, measured in user coordinates
`XN,YN`	The corresponding screen coordinates
`XP,YP`	The screen coordinates of the previous point
`OP%`	The option that a user specifies to control the further action after a polynomial has been drawn
`XS,YS`	The x and y screen coordinates of the y and x axes, respectively
`NN`	The index of the tick marks in the loop that draws and labels them
`CH$`	The character that is being drawn currently during the cycle to display a string

Most of the REM statements in the listing were added after the program was written. Omit every REM statement that is on the same line as another statement, and omit the actual remark in every REM statement that is on a line by itself when you type the program—they increase its length to an extent that makes it inoperable. The null REM statements are needed in case other statements branch to them. To ensure the correctness of the version in this section a simple editing program was used to delete the REM statements and rewrite the shorter version, which was tested again.

The POLYPLOT program first reads the parameters for the preset polynomials from the DATA statements into the appropriate arrays. The main menu is displayed. Depending on the response by the user, the parameters of a preset polynomial are assigned to the variables for current use or the user is prompted for new values. In both cases the program then draws the equation on the graphics area of the screen. Use is made of a set of simple subroutines consisting of HPLOT statements that contain the explicit relative coordinates of the ends of the vectors that form the individual characters. This is a convenient alternative to the FONT FORMER approach when only a few different characters are needed.

Simple tests avoid the inclusion of zero terms, the plus sign at the start of an expression, and unit coefficients and exponents. Floating point fractions are converted to fixed point form. The STR$ function converts numbers to string representations that can be manipulated using the substring functions.

The program draws axes that cross at the center of the portion of the screen that occupies pixel positions 40–240 horizontally and 0–150 vertically. Tick marks are placed at multiples of XD and YD from the origin and labeled using the string drawing tactics just mentioned. The scale factors SX and SY to convert user to screen coordinates are found by use of simple proportionality.

The polynomial is evaluated by the standard algorithm that starts with the highest coefficient, multiplies this by the value of *x*, adds the next coefficient, multiplies the growing result by *x*, and so on, until the constant has been added. Successive points on the curve are joined by straight lines. Points that are out of range are omitted.

Suggested variations

1. After a polynomial has been drawn, allow the option of keeping it on the screen and drawing the next polynomial on the same axes.
2. Allow the axes to be displayed or suppressed optionally.
3. Shade the area between two polynomials.
4. Alter the prompt for new data to allow the user to define the data completely or to specify just the items that are changed from the previous graph and the new values.
5. Alter the program to plot curves that are defined by equations for *x* as polynomials in *y*.

The POLYPLOT program. Note: omit REMs as explained on page 124.

```
100  REM *** POLYPLOT ***
110  LOMEM: 16384: REM *** PROTECT MEMORY ***
120  DIM C0(10),C1(10),C2(10),AX(10),BY(10),D(10): REM *** FOR QUADRATICS ***
130  DIM H0(10),H1(10),H2(10),H3(10),EX(10),FY(10),G(10): REM *** FOR CUBICS ***
140  DIM JX(10),KY(10),GG(10),CC(100): REM *** FOR HIGHER DEGREE POLYNOMIALS ***
150  DIM CD(10): REM *** FOR POLYNOMIAL BEING PLOTTED ***
160  READ IM(1): REM *** NO. OF PRESET QUADRATICS ***
170  FOR IQ=1 TO IM(1): REM *** STORE THE DATA FOR THEM ***
180  READ C0(IQ),C1(IQ),C2(IQ),AX(IQ),BY(IQ),D(IQ)
190  NEXT IQ
200  REM *** DATA FOR QUADRATICS ***
210  DATA 5,3,2,.1,12,20,1,1,-2,-1,2,10,.1,-4,.1,.16,2.5,1.5,.1
220  DATA -3,-18,-20,.25,.5,.01,3,.2,.4,1.25,2.5,.05
230  READ IM(2): REM *** NO. OF PRESET CUBICS ***
240  FOR IQ=1 TO IM(2): REM *** STORE THE DATA FOR THEM ***
250  READ H0(IQ),H1(IQ),H2(IQ),H3(IQ),EX(IQ),FY(IQ),G(IQ)
260  NEXT IQ
270  REM *** DATA FOR CUBICS ***
280  DATA 3,1.5,2,-.06,-.05,2.5,2.5,.1,-5,-6,3,17,.25,2.5,.1
290  DATA 0,-3,1,.5,1,2.5,.1
300  READ IM(3): REM *** NO. OF PRESET HIGHER DEGREE POLYNOMIALS ***
310  IR=0
320  FOR IQ=1 TO IM(3): REM *** STORE THE DATA FOR THEM ***
330  READ DG%:IR=IR+1:CC(IR)=DG%: REM *** STORE THE DEGREE ***
340  FOR IS=0 TO DG%: REM *** STORE THE COEFFICIENTS ***
350  IR=IR+1: READ CC(IR)
360  NEXT IS
370  READ JX(IQ),KY(IQ),GG(IQ): REM *** STORE THE RANGES AND INTERVALS ***
```

```
380  NEXT IQ
390  REM *** DATA FOR HIGHER DEGREE POLYNOMIALS ***
400  DATA 3,5,14, 16,.4,.9,-.01,-.01,2.5,25,.1
410  DATA 6,30,2.25,1.75,-.45,-.1,.007,.001,3,25,.1
420  DATA 6,-6,5.8,25,-83,-137,166,217,.25,5,.01
430  REM *** INITIALIZE THE CONSTANTS ***
440  WD$(0)="STRAIGHT LINE"
450  WD$(1)="QUADRATIC":WD$(2)="CUBIC":WD$(3)="POLYNOMIAL"
460  W=5:H=6:HH=INT(H/2)
470  HGR: HCOLOR=3: REM *** HI-RES GRAPHICS ***
480  VTAB 21: CALL -958: REM *** START AND CLEAR TEXT AREA ***
490  PRINT "TYPE 1 FOR SAMPLE POLYNOMIALS,"
500  PRINT " 2 TO SPECIFY A POLYNOMIAL,"
510  VTAB 23: CALL -958: REM *** START AND CLEAR THE LINE ***
520  INPUT " 3 TO QUIT: ";OP
530  ON OP GOTO 550,830,3000: REM *** BRANCH ON THE RESPONSE ***
540  PRINT "INVALID OPTION": GOTO 510
550  VTAB 21: CALL -958: REM *** START AND CLEAR TEXT AREA ***
560  PRINT "TYPE 1 FOR QUADRATICS, 2 FOR CUBICS,"
570  INPUT " 3 FOR OTHERS, 4 FOR MAIN OPTIONS: ";OQ
580  IF OQ=4 THEN 490: REM *** GO BACK TO MAIN MENU ***
590  IF OQ>0 AND OQ<4 THEN 620
600  PRINT "INVALID REQUEST": GOTO 550
610  REM *** USE PRESET DATA ***
620  IQ=0: REM *** INITIALIZE THE POINTER ***
630  IF IQ>=IM(OQ) THEN IQ=0: REM *** RESTART THE CYCLE IF IT HAS ENDED ***
640  IQ=IQ+1: REM *** MOVE TO THE NEXT POLYNOMIAL ***
650  ON OQ GOTO 660,700,740: REM *** BRANCH ON DEGREE ***
660  DG%=2: REM *** QUADRATIC-COPY THE PARAMETERS USED IN THE PLOTTING ***
670  CD(0)=C0(IQ):CD(1)=C1(IQ):CD(2)=C2(IQ)
680  XD=AX(IQ):YD=BY(IQ):DP=D(IQ)
690  GOTO 1110: REM *** PROCEED TO PLOT ***
700  DG%=3: REM *** CUBIC-COPY THE PARAMETERS USED IN PLOTTING ***
710  CD(0)=H0(IQ):CD(1)=H1(IQ):CD(2)=H2(IQ):CD(3)=H3(IQ)
720  XD=EX(IQ):YD=FY(IQ):DP=G(IQ)
730  GOTO 1110: REM *** PROCEED TO PLOT ***
740  IF IQ=1 THEN IR=0: REM *** GENERAL DEGREE-RESET POINTER AT START OF CYCLE ***
750  IR=IR+1
760  DG%=CC(IR): REM *** COPY THE DEGREE ***
770  FOR IS=0 TO DG%: REM ** COPY THE COEFFICIENTS ***
780  IR=IR+1:CD(IS)=CC(IR)
790  NEXT IS
800  XD=JX(IQ):YD=KY(IQ):DP=GG(IQ): REM *** COPY THE RANGE AND INTERVAL ***
810  GOTO 1110: REM *** PROCEED TO PLOT ***
820  REM *** POLYNOMIAL SPECS WILL BE TYPED AS INPUT ***
830  HGR: HCOLOR=3: VTAB 21: CALL -958: REM *** GRAPHICS & TEXT AREA CLEARED ***
840  INPUT "DEGREE: ";DG%
850  OQ=0: IF DG%<2 THEN 890: REM *** SET SWITCH FOR DESCRIPTIVE WORD ***
860  IF DG%=2 THEN OQ=1: GOTO 890
```

```
 870  IF DG%=3 THEN OQ=2: GOTO 890
 880  OQ=3
 890  FOR TN=0 TO DG%: REM *** PROMPT FOR COEFFICIENTS ***
 900  VTAB 22: CALL -958
 910  IF TN>0 THEN 930
 920  INPUT "CONSTANT=";CD(0): GOTO 970
 930  IF TN>1 THEN 950
 940  INPUT "COEFFT OF X=";CD(1): GOTO 970
 950  PRINT "COEFFT OF X TO POWER ";TN;"=";
 960  INPUT "";CD(TN)
 970  NEXT TN: REM *** END OF COEFFICIENT INPUT CYCLE ***
 980  VTAB 22: CALL -958: REM *** PROMPT FOR FURTHER DATA ***
 990  INPUT "TICK MARK INTERVALS: ";XD,YD
1000  INPUT "X INTERVAL FOR PLOTTING: ";DP
1100  REM *** PLOT A POLYNOMIAL ***
1110  XL=0:YL=154: REM *** POSITION WHERE EQUATION WILL BE DISPLAYED ***
1120  HGR: HCOLOR=3
1130  VTAB 21: CALL -958: PRINT : REM *** CLEAR THE LINE ***
1140  FLASH : HTAB 10: PRINT "PLOTTING A ";WD$(OQ): NORMAL
1150  GOSUB 5390: REM *** DISPLAY Y ***
1160  GOSUB 5370: REM *** DISPLAY = ***
1170  SW%=0: REM *** SWITCH TO SUPRESS LEADING + SIGN ***
1180  XB%=XL: REM *** SET THE BACKTRACK POSITION ***
1190  FOR NT=0 TO DG%: REM *** CYCLE THROUGH COEFFICIENTS ***
1200  IF XL+3*W<278 THEN 1220: REM *** BRANCH IF THERE IS ENOUGH ROOM ***
1210  GOSUB 4800: GOTO 1600: REM *** ERASE ANY OF THE TERM DRAWN ALREADY ***
1220  XB%=XL: REM *** SET THE BACKTRACK POSITION ***
1230  IF CD(NT)=0 THEN 1540: REM *** BRANCH IF THE COEFFICIENT IS ZERO ***
1240  IF SW%<>0 AND CD(NT)>0 THEN  GOSUB 5350: REM *** DRAW A + ***
1250  IF CD(NT)<0 THEN  GOSUB 5330: REM *** DRAW A - ***
1260  ST$=STR$(ABS(CD(NT))):REM *** PREPARE TO DRAW THE COEFFICIENT ***
1270  IF NT>0 AND ABS(CD(NT))=1 THEN 1480: REM *** BRANCH IF COEFFT. IS 1 OR -1 ***
1280  FOR I=1 TO LEN(ST$)
1290  IF MID$(ST$,I,1)="E" THEN 1320: REM *** BRANCH IF FLOATING POINT ***
1300  NEXT I
1310  GOTO 1400: REM *** BRANCH IF IT IS IN FIXED POINT ***
1320  EX=VAL(MID$(ST$,I+1)): REM *** EXTRACT EXPONENT OF FLOATING POINT NO. ***
1330  IF EX>=0 THEN 1400: REM *** BRANCH IF THE EXPONENT IS POSITIVE ***
1340  MA=INT(VAL(LEFT$(ST$,I-1))+.5): REM *** EXTRACT THE MANTISSA ***
1350  ST$=".": REM *** PROVIDE A PERIOD ***
1360  FOR II=-2 TO EX STEP -1: REM *** APPEND THE NECESSARY NO. OF ZEROS ***
1370  ST$=ST$+"0"
1380  NEXT II
1390  ST$=ST$+STR$(MA): REM *** APPEND THE SIGNIFICANT DIGITS ***
1400  GOSUB 4500: REM *** DRAW THE STRING ***
1480  IF NT>0 THEN  GOSUB 5420: REM *** DRAW THE X WHEN NEEDED ***
1490  IF NT<2 THEN 1530: REM *** BRANCH IF THE EXPONENT IS NOT TO BE DRAWN ***
1500  YL=YL-HH:ST$=STR$(NT): REM *** RAISE THE PLOTTING POSITION ***
1510  GOSUB 4500:REM *** DRAW THE EXPONENT ***
```

```
1520  YL=YL+HH: REM *** RESTORE THE VERTICAL PLOTTING POSITION ***
1530  SW%=1: REM *** SHOWS THAT A TERM HAS BEEN DRAWN ***
1540  NEXT NT: REM *** END OF CYCLE TO DRAW SUCCESSIVE TERMS ***
1600  GOSUB 4000: REM *** BRANCH TO PLOT AND LABEL THE AXES ***
1610  SX=24/XD:SY=24/YD: REM *** FACTORS TO GO FROM USER TO SCREEN COORDS ***
1620  FOR XU=-4*XD TO 4*XD STEP DP: REM *** CYCLE TO COMPUTE SUCCESSIVE VALUES ***
1630  YU=CD(DG%): REM *** START OF POLYNOMIAL EVALUATION ***
1640  IF DG%=0 THEN YU=CD(0): GOTO 1680: REM *** DEGREE IS ZERO ***
1650  FOR TN=DG%-1 TO 0 STEP -1: REM *** CYCLE TO EVALUATE A POLYNOMIAL ***
1660  YU=YU*XU+CD(TN)
1670  NEXT TN
1680  XN=140+SX*XU:YN=72-SY*YU: REM *** CONVERT TO SCREEN COORDS ***
1690  IF XU=-4*XD THEN 1720: REM *** BRANCH AT START OF PLOT ***
1700  IF YN<0 OR 144<YN OR YP<0 OR 144<YP THEN 1720: REM ***OUT OF RANGE ***
1710  HPLOT XP,YP TO XN,YN: REM *** DRAW FROM THE PREVIOUS COMPUTED POINT ***
1720  XP=XN:YP=YN: REM *** SET THE "PREVIOUS COORDS" FOR NEXT CYCLE ***
1730  NEXT XU: REM *** END OF CYCLE TO COMPUTE SUCCESSIVE VALUES ***
1740  VTAB 21: CALL -958: REM *** PREPARE TO PROMPT FOR NEXT ACTION ***
1750  IF OP<>1 THEN 1810: REM *** BRANCH IF USER IS SPECIFYING POLYNOMIALS ***
1760  VTAB 22: PRINT "TYPE 1 FOR ANOTHER ";WD$(OQ)
1770  INPUT " 2 TO RETURN TO THE MAIN MENU: ";OP%
1780  ON OP% GOTO 630,470
1790  PRINT "INVALID REQUEST-PRESS A KEY"
1800  GET A$: GOTO 1740
1810  VTAB 22: PRINT "TYPE 1 TO SPECIFY ANOTHER POLYNOMIAL,"
1820  INPUT " 2 TO RETURN TO TO THE MAIN MENU: ";OP%
1830  ON OP% GOTO 830,470
1840  PRINT "INVALID REQUEST-PRESS A KEY"
1850  GET A$: GOTO 1810
3000  HOME : END
4000  REM *** PLOT AND LABEL THE AXES ***
4010  HPLOT 42,72 TO 238,72: REM *** X AXIS ***
4020  HPLOT 140,0 TO 140,144: REM *** Y AXIS ***
4030  XU=-4*XD:XS=40: REM *** PREPARE TO DRAW TICK MARKS ON X AXIS ***
4040  FOR N=-4 TO 4
4050  IF N=0 THEN 4080
4060  HPLOT XS,72 TO XS,67
4070  ST$=STR$(XU):XL=XS+2:YL=72+H: GOSUB 4500: REM *** DRAW THE LABEL ***
4080  XS=XS+25:XU=XU+XD
4090  NEXT N
4100  YU=-3*YD:YS=144: REM *** PREPARE TO DRAW TICK MARKS ON Y AXIS ***
4110  FOR N=-3 TO 3
4120  IF N=0 THEN 4150
4130  HPLOT 140,YS TO 145,YS
4140  ST$=STR$(YU):XL=138-W*LEN(ST$):YL=YS: GOSUB 4500
4150  YU=YU+YD:YS=YS-24
4160  NEXT N
4170  RETURN
4500  REM *** DISPLAY A STRING ***
```

```
4510  FOR I=1 TO LEN(ST$): REM *** CYCLE THROUGH SUCCESSIVE CHARACTERS ***
4520  IF XL+W>=279 THEN  GOSUB 4800: RETURN : REM *** BRANCH IF TOO LONG ***
4530  CH$=MID$(ST$,I,1): REM *** EXTRACT A CHARACTER ***
4540  IF CH$="0" THEN  GOSUB 5010
4550  IF CH$="1" THEN  GOSUB 5030
4560  IF CH$="2" THEN  GOSUB 5060
4570  IF CH$="3" THEN  GOSUB 5100
4580  IF CH$="4" THEN  GOSUB 5140
4590  IF CH$="5" THEN  GOSUB 5170
4600  IF CH$="6" THEN  GOSUB 5200
4610  IF CH$="7" THEN  GOSUB 5220
4620  IF CH$="8" THEN  GOSUB 5240
4630  IF CH$="9" THEN  GOSUB 5280
4640  IF CH$="." THEN  GOSUB 5310
4650  IF CH$="-" THEN  GOSUB 5330
4660  IF CH$="E" THEN  GOSUB 5440
4750  NEXT I
4760  RETURN
4800  HCOLOR=0: REM *** ERASE FROM BACKTRACK TO CURRENT X POSITION ***
4810  FOR YY=154-HH TO 154+H
4820  HPLOT XB%,YY TO XL,YY
4830  NEXT YY
4840  XL=XB%: HCOLOR=3
4850  GOSUB 5100: GOSUB 5100: GOSUB 5100
4860  RETURN
5000  REM *** PLOT 0 THROUGH 9 ***
5010  HPLOT XL,YL TO XL+2,YL TO XL+2,YL+4 TO XL,YL+4 TO XL,YL
5020  XL=XL+W: RETURN
5030  HPLOT XL,YL+1 TO XL+1,YL TO XL+1,YL+4
5040  HPLOT XL,YL+4 TO XL+2,YL+4
5050  XL=XL+W: RETURN
5060  HPLOT XL,YL TO XL+1,YL
5070  HPLOT XL+2,YL+1 TO XL,YL+3
5080  HPLOT XL,YL+4 TO XL+2,YL+4
5090  XL=XL+W: RETURN
5100  HPLOT XL,YL TO XL+1,YL: HPLOT XL+2,YL+1
5110  HPLOT XL+1,YL+2: HPLOT XL+2,YL+3
5120  HPLOT XL,YL+4 TO XL+2,YL+4
5130  XL=XL+W: RETURN
5140  HPLOT XL,YL TO XL,YL+2 TO XL+2,YL+2
5150  HPLOT XL+2,YL TO XL+2,YL+4
5160  XL=XL+W: RETURN
5170  HPLOT XL+2,YL TO XL,YL TO XL,YL+2 TO XL+1,YL+2
5180  HPLOT XL,YL+4 TO XL+1,YL+4: HPLOT XL+2,YL+3
5190  XL=XL+W: RETURN
5200  HPLOT XL,YL TO XL,YL+4 TO XL+2,YL+4 TO XL+2,YL+2 TO XL,YL+2
5210  XL=XL+W: RETURN
5220  HPLOT XL,YL TO XL+2,YL TO XL,YL+4
5230  XL=XL+W: RETURN
```

```
5240  HPLOT XL+1,YL TO XL+2,YL TO XL+2,YL+1
5250  HPLOT XL,YL+1: HPLOT XL+1,YL+2: HPLOT XL,YL+4
5260  HPLOT XL+1,YL+4 TO XL+2,YL+4 TO XL+2,YL+3
5270  XL=XL+W: RETURN
5280  HPLOT XL,YL+4 TO XL+2,YL+4 TO XL+2,YL TO XL,YL TO XL,YL+2 TO XL+2,YL+2
5290  XL=XL+W: RETURN
5300  REM *** PLOT .-+=YXE ***
5310  HPLOT XL+2,YL+4
5320  XL=XL+W: RETURN
5330  HPLOT XL,YL+2 TO XL+2,YL+2
5340  XL=XL+W: RETURN
5350  HPLOT XL,YL+2 TO XL+2,YL+2: HPLOT XL+1,YL+1 TO XL+1,YL+3
5360  XL=XL+W: RETURN
5370  HPLOT XL+1,YL+1 TO XL+3,YL+1: HPLOT XL+1,YL+3 TO XL+3,YL+3
5380  XL=XL+W: RETURN
5390  HPLOT XL,YL TO XL+2,YL+2 TO XL+4,YL
5400  HPLOT XL+2,YL+4 TO XL+2,YL+2
5410  XL=XL+W: RETURN
5420  HPLOT XL,YL+4 TO XL+4,YL: HPLOT XL,YL TO XL+4,YL+4
5430  XL=XL+W+2: RETURN
5440  HPLOT XL+2,YL TO XL,YL TO XL,YL+4 TO XL+2,YL+4: HPLOT XL,YL+2 TO XL+2,YL+2
5450  XL=XL+W+1: RETURN
```

6.3 MAKING A POLYNOMIAL FIT

Just as you can draw a straight line through any two points, you can draw a quadratic curve through any three points with different values of x. You can also draw a cubic curve through any four points, a quartic curve (highest power of x is 4) through any five points, a quintic curve (highest power of x is 5) through any six points, and so on, when the points all have different x values.

The program POLYFIT prompts for the ranges that the x and y axes should span, the number of points, and their coordinates. It then draws the axes, marks the points, draws the appropriate polynomial through them, and displays the equation at the bottom of the screen (Figure 6.9). The program then pauses, having performed the longest calculation needed in this book, if there were many points to be fitted.

The program is a little sensitive to points that are very close together—this can make it try dividing two numbers that are both close to zero, with a result that is not very accurate. So please be kind to it. The time it takes to compute the coefficients goes up as the square of the number of points, and the accuracy goes down as this number goes up.

This is a good example of the kind of program that uses mathematics that most users, including scientists and engineers, take for granted. The problem is easy to understand—you probably solved pairs of simultaneous equations in high school algebra. You might even have tried to solve 3 equations for 3 unknowns, but very few people try to solve more these days using pencil and paper. To fit a 10th-degree polynomial to 11 points involves solving a set of 11 simultaneous equations that contain 11 unknowns. This is a problem in linear algebra that involves a process called matrix inversion.

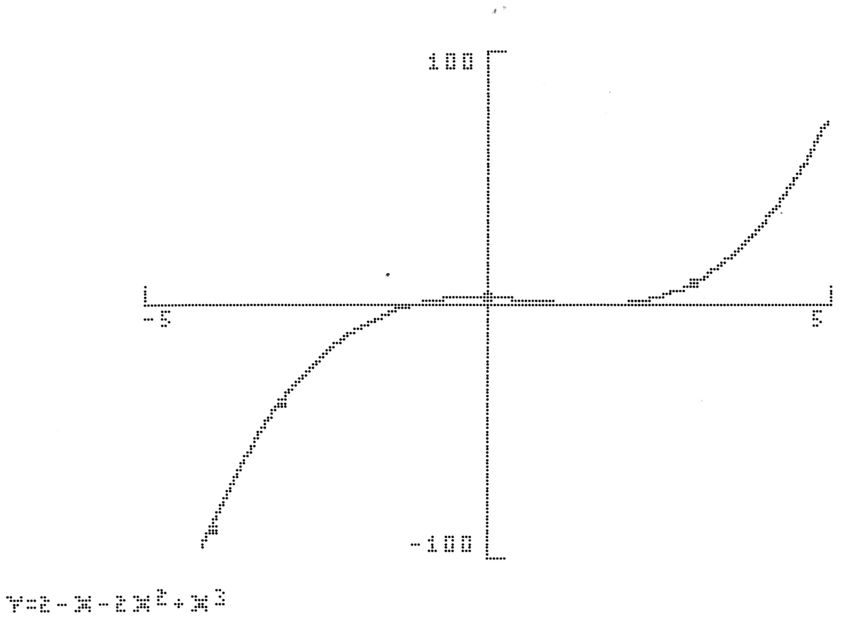


Figure 6.9

POLYFIT uses a very simple method for this, called Gauss-Jordan elimination, which is easy to program, but sensitive to awkward data. Much better methods have been developed in the last few decades, spurred on by the fact that computers could be used to solve problems that could not be tackled before. A considerable amount of expert knowledge and experience has gone into the development of very powerful and dependable mathematical packages for large computers. The time is ripe for corresponding packages for the Apple.

It is very easy to construct the simultaneous equations that define the polynomial that fits a set of n points. If a_0, a_1, ..., $a_{n\text{-}1}$ stand for the coefficients, and x_i, y_i stand for the coordinates of the ith point to be fitted, then

$$y_i = a_0 + a_1x_i + a_2x_i^{\,2} + \ldots + a_{n-1}x_i^{\,n-1}$$

The n equations in which i takes values $1-n$ determine the n unknowns a_0, a_1, ..., $a_{n\text{-}1}$. The coefficients of these unknowns in the ith equation are just the successive powers of x_i from 1 to $x_i^{\,n-1}$.

To explain the method of solution, the equations are rewritten with the unknowns denoted by u_k, the values on the right-hand sides by v_j, and the coefficients by $c_{j,k}$, with j and k taking values $1-n$.

$$c_{1,1}u_1 + c_{1,2}u_2 + \ldots + c_{1,n}u_n = v_1$$

$$c_{2,1}u_1 + c_{2,2}u_2 + \ldots + c_{2,n}u_n = v_2$$

$$\begin{array}{c}\cdot\\\cdot\\\cdot\end{array}$$

$$c_{n,1}u_1 + c_{n,2}u_2 + \ldots + c_{n,n}u_n = v_n$$

The calculation goes through n cycles because there are n equations in n unknowns. In the first cycle the first equation is divided by $c_{1,1}$. The coefficients $c_{1,2}$ to $c_{1,n}$ are replaced by $c_{1,2}/c_{1,1}$ to $c_{1,n}/c_{1,1}$. The right-hand side is replaced by $v_1/c_{1,1}$. The coefficient $c_{1,1}$ is replaced by 1. In the second equation the coefficient $c_{2,2}$ is replaced by $c_{2,2}-c_{2,1}c_{1,2}$. The coefficient $c_{2,3}$ is replaced by $c_{2,3}-c_{2,1}c_{1,3}$ and so on. The right-hand side is replaced by $v_2-c_{2,1}v_1$ and the coefficient $c_{2,1}$ is replaced by 0. These steps use the new values of the $c_{1,j}$ and v_1 in the first equation.

The further equations are treated similarly. Each coefficient $c_{j,k}$ is replaced by $c_{j,k}-c_{j,1}c_{1,k}$. Each right-hand side v_j is replaced by $v_j-c_{j,1}v_1$. Each coefficient in the first column is replaced by 0. This column now has 1 in the first row and 0 elsewhere.

The second equation plays the same role in the second cycle that the first equation plays in the first cycle. The equation is divided by the element $c_{2,2}$. Each of the coefficients $c_{j,k}$ on the right of the diagonal in the other equations is replaced by $c_{j,k}-c_{j,2}c_{2,k}$ and each right-hand side v_j is replaced by $v_j-c_{j,2}v_2$ At the end of this cycle the coefficients $c_{1,1}$ and $c_{2,2}$ are 1 and all the other coefficients in the first two columns are 0.

The third equation plays the corresponding role in the third cycle, the fourth equation in the fourth cycle, and so on until the end. When the third cycle is complete the diagonal coefficients $c_{1,1}$, $c_{2,2}$ and $c_{3,3}$ are 1 and all the other coefficients in the first three columns are 0. At the end of the jth cycle the diagonal coefficients in the first j columns are 1 and all the other coefficients in these columns are 0. At the end of the nth cycle all the diagonal coefficients are 1 and all the other coefficients are 0. The first equation just has 1 times u_1 on the left —all the other terms in it are 0 because all the other coefficients are 0. The second equation just has 1 times u_2 on the left, correspondingly. The right-hand side of the jth equation at the end of the nth cycle is thus the value of the jth unknown.

You can verify this, performing the calculation by hand with two or even three or four equations that you know will have simple solutions. You can ensure this by first constructing the equations from simple values of the quantities that you will then treat as unknowns. For example, if you use the values 2, 5, and 1 for u_1, u_2, and u_3, the equations

$$3u_1+u_2+u_3=12 \qquad u_1+2u_2-u_3=11 \qquad u_1+u_2+3u_3=10$$

will lead you back to 2, 5, and 1.

The main variables in the program are as follows

`W,H`	The width and height of the characters
`HH`	Half the height of the characters
`XP(P)`	The x coordinate of the Pth point
`B(P)`	The y coordinate of the Pth point
`CD(NT)`	The Nth coefficient in the polynomial
`A(R,Q)`	The (R, Q)th element of the matrix that is being inverted Initially this is XP(R) raised to the power Q−1
`XV,XH`	The low and high values of x in user coordinates
`YV,YH`	The low and high values of y in user coordinates
`XA`	The x coordinate of the y axis in screen coordinates
`YA`	The y coordinate of the x axis in screen coordinates
`XL,YL`	The screen coordinates of the beginning of a string
`ST$`	The string to be displayed
`SX,SY`	The scaling factor to convert user coordinates to screen coordinates

D	The number of points to be fitted by the polynomial
P	The index that identifies the points that the user specifies, the rows of the matrix that is being inverted, and the coefficients in the polynomial that results
XN,YN	The screen coordinates of a point through which the polynomial must pass
Q	The index that identifies the columns of the matrix and, in particular, the powers of x when the matrix is formed
R	The index of the rows of the matrix in the inner cycle during inversion
DV	The current diagonal element being divided into the elements in lower rows of the matrix
MY	The current element used as the multiplier in the reduction of the matrix
OP	The option that determines the action after a polynomial has been fitted

The POLYFIT program follows the mathematical process in a direct fashion. It also uses the same tactics as POLYPLOT to draw equations on the screen, to plot, and to label axes and to evaluate a polynomial from its coefficients.

Suggested variations

1. Fit a polynomial for x in terms of y to points with different y values.

The POLYFIT program. Note: omit REMs as explained on page 124.

```
100  REM *** POLYFIT-FITS POLYNOMIALS ***
110  LOMEM: 16384: REM *** PROTECTS MEMORY ***
120  XI=.1:W=5:H=4:HH=2: REM *** PLOTTING INTERVAL & FONT DIMENSIONS ***
130  DIM CD(10),XP(10),A(10,10),B(10): REM *** ARRAYS FOR COMPUTATION ***
140  TEXT : HOME : HGR: HCOLOR=3
150  POKE 34,22: HOME : REM *** LIMITS TEXT TO TWO BOTTOM LINES ***
160  INPUT "X LIMITS: ";XV,XH
170  INPUT "Y LIMITS: ";YV,YH
180  REM *** PLOT THE AXES ***
190  IF XV<=0 AND XH>=0 THEN XA=40-200*XV/(XH-XV): REM *** X BOTH + & - ***
200  IF XV>0 OR XH<0 THEN XA=40: REM *** X ENTIRELY + OR ENTIRELY - ***
210  HPLOT XA,0 TO XA,144: REM *** PLOT THE Y AXIS ***
220  IF YV<=0 AND YH>=0 THEN YA=144*YH/(YH-YV): REM *** Y BOTH + & - ***
230  IF YV>0 OR YH<0 THEN YA=144: REM *** Y ENTIRELY + OR ENTIRELY - ***
235  IF YA<10 THEN YA=144: REM *** ALLOW SPACE FOR TICK MARKS ***
240  HPLOT 40,YA TO 240,YA: REM *** PLOT THE X AXIS ***
250  HPLOT XA,0 TO XA+5,0: REM *** PLOT TICK MARK AT TOP OF Y AXIS ***
260  ST$=STR$(YH): REM *** LABEL IT ***
270  XL=XA-W*LEN(ST$)-2:YL=1: GOSUB 4500
280  HPLOT XA,144 TO XA+5,144: REM *** PLOT TICK MARK AT BOTTOM OF Y AXIS ***
290  ST$=STR$(YV): REM *** LABEL IT ***
300  XL=XA-W*LEN(ST$)-2:YL=142-H: GOSUB 4500
```

```
310  HPLOT 40,YA TO 40,YA-5: REM *** PLOT TICK MARK AT LEFT OF X AXIS ***
320  ST$=STR$(XV): REM *** LABEL IT ***
330  XL=40:YL=YA+2: GOSUB 4500
340  HPLOT 240,YA TO 240,YA-5: REM *** PLOT TICK MARK AT RIGHT OF IT ***
350  ST$=STR$(XH): REM *** LABEL IT ***
360  XL=240-W*LEN(ST$):YL=YA+2: GOSUB 4500
370  SX=200/(XH-XV):SY=144/(YH-YV): REM *** SCALING FACTOR FOR COORDS ***
380  HOME : INPUT "NUMBER OF POINTS: ";D
390  IF D>1 AND D<11 THEN 420: REM *** BRANCH IF NUMBER OF POINTS IS VALID ***
400  FLASH : PRINT "INVALID NUMBER-PRESS A KEY": NORMAL : GET A$: GOTO 380
410  POKE 34,23: REM *** PROTECT LINE 23 ON THE SCREEN ***
420  FOR P=1 TO D: REM *** CYCLE TO STORE THE SUCCESSIVE POINTS ***
430  HOME : INPUT "COORDINATES: ";XP(P),B(P)
440  A(P,1)=1
450  FOR Q=2 TO D: REM *** STORE THE POWERS OF X IN THE A MATRIX ***
460  A(P,Q)=XP(P)*A(P,Q-1)
470  NEXT Q
480  XN=40+SX*(XP(P)-XV): REM *** FORM THE SCREEN COORDS FOR NEW POINT ***
490  YN=144-SY*(B(P)-YV)
500  IF 40>XN OR 240<XN OR 0>YN OR 144<YN THEN 540: REM *** NOT PLOTTABLE ***
510  HPLOT XN,YN: REM *** PLOT THE POINT AND A SQUARE AROUND IT IF POSSIBLE ***
520  IF YN>0 AND XN>40 AND XN<240 THEN  HPLOT XN-1,YN TO XN-1,YN-1 TO XN+1,YN-1 TO
      XN+1,YN
530  IF YN<144 AND XN>40 AND XN<240 THEN HPLOT XN-1,YN TO XN-1,YN+1 TO XN+1,YN+1 T
     O XN+1,YN
540  HOME : NEXT P: REM *** END OF CYCLE TO STORE THE SUCCESSIVE POINTS ***
550  REM *** SOLVE THE EQUATIONS ***
560  POKE 34,22:HOME:HTAB 10:FLASH:PRINT "COMPUTING THE COEFFICIENTS":NORMAL
570  FOR P=1 TO D: REM *** PERFORM THE MAIN ITERATION D TIMES ***
580  IF A(P,P)=1 THEN 650: REM *** BRANCH IF DIAGONAL ELEMENT IS 1 ALREADY ***
590  DV=A(P,P):A(P,P)=1: REM *** DIVIDE THE ROWS BELOW IT BY IT ***
600  IF P=D THEN 640: REM *** BRANCH IF AT THE BOTTOM ROW ALREADY ***
610  FOR Q=P+1 TO D: REM *** START OF CYCLE TO DIVIDE LOWER ROWS ***
620  A(P,Q)=A(P,Q)/DV
630  NEXT Q: REM *** END OF CYCLE TO DIVIDE THE LOWER ROWS ***
640  B(P)=B(P)/DV: REM *** DIVIDE THE P-TH ELEMENT IN THE RESULT VECTOR ***
650  FOR R=1 TO D: REM *** START OF OUTER CYCLE TO REDUCE ELEMENTS ***
660  IF R=P THEN 740: REM *** BYPASS ACTIONS FOR P-TH ROW ***
670  MY=A(R,P): REM *** SELECT ELEMENT FROM R-TH ROW ***
680  A(R,P)=0
690  IF P=D THEN 730
700  FOR Q=P+1 TO D: REM *** START OF INNER CYCLE FOR RIGHT OF DIAGONAL ***
710  A(R,Q)=A(R,Q)-MY*A(P,Q)
720  NEXT Q: REM *** END OF INNER CYCLE FOR RIGHT OF DIAGONAL ELEMENTS ***
730  B(R)=B(R)-MY*B(P): REM *** REDUCE THE R-TH ELEMENT IN THE RESULT VECTOR ***
740  NEXT R: REM *** END OF OUTER CYCLE TO REDUCE ELEMENTS ***
750  NEXT P: REM *** END OF CYCLE FOR MAIN ITERATION ***
760  FOR TN=0 TO D-1: REM *** TRANSFER RESULTS FOR CONSISTENCY WITH POLYPLOT ***
770  CD(TN)=B(TN+1)
```

```
 780  NEXT TN
 790  DG%=D-1
 800  HOME : REM *** CLEAR THE TEXT AREA ***
1100  REM *** PLOT THE POLYNOMIAL-BEGIN BY DRAWING THE EQUATION (SEE POLYPLOT) ***
1110  XL=0:YL=155
1150  GOSUB 5390: REM *** DISPLAY Y ***
1160  GOSUB 5370: REM *** DISPLAY = ***
1170  SW%=0
1180  XB%=XL
1190  FOR NT=0 TO DG%
1200  IF XL+3*W<278 THEN 1220
1210  GOSUB 4700: GOTO 1600
1220  XB%=XL
1230  IF CD(NT)=0 THEN 1540
1240  IF SW%<>0 AND CD(NT)>0 THEN  GOSUB 5350: REM *** + ***
1250  IF CD(NT)<0 THEN  GOSUB 5330: REM *** - ***
1260  ST$=STR$(ABS(CD(NT)))
1270  IF NT>0 AND ABS(CD(NT))=1 THEN 1480
1280  FOR I=1 TO LEN(ST$)
1290  IF MID$(ST$,I,1)="E" THEN 1320
1300  NEXT I
1310  GOTO 1410
1320  EX=VAL(MID$(ST$,I+1))
1330  IF EX>=0 THEN 1470
1340  MA=INT(VAL(LEFT$(ST$,I-1))+.5)
1350  ST$="."
1360  FOR II=-2 TO EX STEP -1
1370  ST$=ST$+"0"
1380  NEXT II
1390  ST$=ST$+STR$(MA)
1400  GOTO 1470
1410  IF LEN(ST$)<6 THEN 1470: REM *** BRANCH IF # LESS THAN 6 DIGITS ***
1420  FOR I=1 TO LEN(ST$): REM *** SCAN FOR DECIMAL POINT ***
1430  IF MID$(ST$,I,1)="." THEN 1460
1440  NEXT I
1450  GOTO 1470
1460  IF I+3<LEN(ST$) THEN ST$=LEFT$(ST$,I+3): REM *** TRUNCATE IF POSSIBLE ***
1470  GOSUB 4500: REM *** COEFFICIENT ***
1480  IF NT>0 THEN  GOSUB 5420: REM *** X ***
1490  IF NT<2 THEN 1530
1500  YL=YL-HH:ST$=STR$(NT)
1510  GOSUB 4500: REM *** EXPONENT ***
1520  YL=YL+HH
1530  SW%=1
1540  NEXT NT
1600  HOME : PRINT : FLASH : HTAB 10: PRINT "PLOTTING THE POLYNOMIAL": NORMAL
1610  FOR XU=XV TO XH STEP XI: REM *** PLOTTING CYCLE ***
1620  YU=CD(DG%): REM *** BEGIN THE EVALUATION ***
1630  IF DG%=0 THEN 1670
```

```
1640  FOR TN=DG%-1 TO 0 STEP -1: REM *** CYCLE FOR SUCCESSIVE COEFFICIENTS ***
1650  YU=YU*XU+CD(TN)
1660  NEXT TN
1670  XN=40+SX*(XU-XV):YN=144-SY*(YU-YV): REM *** SCREEN COORDINATES ***
1680  IF XU=XV THEN 1710
1690  IF YN<0 OR 144<YN OR YP<0 OR 144<YP THEN 1710
1700  HPLOT XP,YP TO XN,YN: REM *** DRAW FROM PREVIOUS POINT ***
1710  XP=XN:YP=YN
1720  NEXT XU
1721  REM *** PROMPT FOR FURTHER ACTION ***
1730  POKE 34,21: HOME : PRINT "TYPE 1 TO LIST COEFFICIENTS,"
1740  PRINT "      2 TO FIT ANOTHER POLYNOMIAL,"
1750  INPUT " 3 TO QUIT: ";OP
1760  ON OP GOTO 1780,140,1840
1770  FLASH : PRINT "INVALID REQUEST": NORMAL : GOTO 1730
1780  POKE 34,0: TEXT : HOME
1790  PRINT "POWER OF X","COEFFICIENT": PRINT
1800  FOR TN=0 TO DG%: REM *** DISPLAY THE COEFFICIENTS ***
1810  PRINT TN,CD(TN)
1820  NEXT TN
1830  GOTO 1730
1840  POKE 34,0: TEXT : HOME : END : REM *** QUIT ***
4500  REM *** DISPLAY A STRING ***
4510  FOR I=1 TO LEN(ST$)
4530  CH$=MID$(ST$,I,1)
4540  IF CH$="0" THEN  GOSUB 5010
4550  IF CH$="1" THEN  GOSUB 5030
4560  IF CH$="2" THEN  GOSUB 5060
4570  IF CH$="3" THEN  GOSUB 5100
4580  IF CH$="4" THEN  GOSUB 5140
4590  IF CH$="5" THEN  GOSUB 5170
4600  IF CH$="6" THEN  GOSUB 5200
4610  IF CH$="7" THEN  GOSUB 5220
4620  IF CH$="8" THEN  GOSUB 5240
4630  IF CH$="9" THEN  GOSUB 5280
4640  IF CH$=" " THEN  GOSUB 5310
4650  IF CH$="-" THEN  GOSUB 5330
4660  IF CH$="E" THEN  GOSUB 5440
4750  NEXT I
4760  RETURN
5000  REM *** PLOT 0 THROUGH 9 ***
5010  HPLOT XL,YL TO XL+2,YL TO XL+2,YL+4 TO XL,YL+4 TO XL,YL
5020  XL=XL+W: RETURN
5030  HPLOT XL,YL+1 TO XL+1,YL TO XL+1,YL+4
5040  HPLOT XL,YL+4 TO XL+2,YL+4
5050  XL=XL+W: RETURN
5060  HPLOT XL,YL TO XL+1,YL
5070  HPLOT XL+2,YL+1 TO XL,YL+3
5080  HPLOT XL,YL+4 TO XL+2,YL+4
```

```
5090  XL=XL+W: RETURN
5100  HPLOT XL,YL TO XL+1,YL: HPLOT XL+2,YL+1
5110  HPLOT XL+1,YL+2: HPLOT XL+2,YL+3
5120  HPLOT XL,YL+4 TO XL+2,YL+4
5130  XL=XL+W: RETURN
5140  HPLOT XL,YL TO XL,YL+2 TO XL+2,YL+2
5150  HPLOT XL+2,YL TO XL+2,YL+4
5160  XL=XL+W: RETURN
5170  HPLOT XL+2,YL TO XL,YL TO XL,YL+2 TO XL+1,YL+2
5180  HPLOT XL,YL+4 TO XL+1,YL+4: HPLOT XL+2,YL+3
5190  XL=XL+W: RETURN
5200  HPLOT XL,YL TO XL,YL+4 TO XL+2,YL+4 TO XL+2,YL+2 TO XL,YL+2
5210  XL=XL+W: RETURN
5220  HPLOT XL,YL TO XL+2,YL TO XL,YL+4
5230  XL=XL+W: RETURN
5240  HPLOT XL+1,YL TO XL+2,YL TO XL+2,YL+1
5250  HPLOT XL,YL+1: HPLOT XL+1,YL+2: HPLOT XL,YL+4
5260  HPLOT XL+1,YL+4 TO XL+2,YL+4 TO XL+2,YL+3
5270  XL=XL+W: RETURN
5280  HPLOT XL,YL+4 TO XL+2,YL+4 TO XL+2,YL TO XL,YL TO XL,YL+2 TO XL+2,YL+2
5290  XL=XL+W: RETURN
5300  REM *** PLOT  -+=YXE ***
5310  HPLOT XL+2,YL+4
5320  XL=XL+W: RETURN
5330  HPLOT XL,YL+2 TO XL+2,YL+2
5340  XL=XL+W: RETURN
5350  HPLOT XL,YL+2 TO XL+2,YL+2: HPLOT XL+1,YL+1 TO XL+1,YL+3
5360  XL=XL+W: RETURN
5370  HPLOT XL+1,YL+1 TO XL+3,YL+1: HPLOT XL+1,YL+3 TO XL+3,YL+3
5380  XL=XL+W: RETURN
5390  HPLOT XL,YL TO XL+2,YL+2 TO XL+4,YL
5400  HPLOT XL+2,YL+4 TO XL+2,YL+2
5410  XL=XL+W: RETURN
5420  HPLOT XL,YL+4 TO XL+4,YL: HPLOT XL,YL TO XL+4,YL+4
5430  XL=XL+W+2: RETURN
5440  HPLOT XL+2,YL TO XL,YL TO XL,YL+4 TO XL+2,YL+4: HPLOT XL,YL+2 TO XL+2,YL+2
5450  XL=XL+W+1: RETURN
```

6.4 SOME CURVES TO SMOOTH OUT ANGLES

Would you like to sketch a head or a car or a wave breaking on a beach, just by plotting a few points that approximate the outline and letting the computer draw a smooth curve within the framework you provided? The smooth outline of a head (Figure 6.11) is produced from the points in the angular drawing (Figure 6.10) by a method that has been used to design automobiles and aeroplane parts for several years, as a major industrial application of computer graphics. Invented by the French engineer Bezier it has up to now needed very expensive equipment. Now you can use it on an Apple.

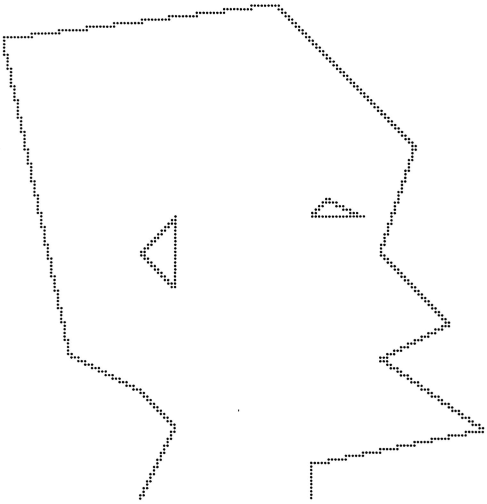


Figure 6.10. "Dot-to-dot" drawing.

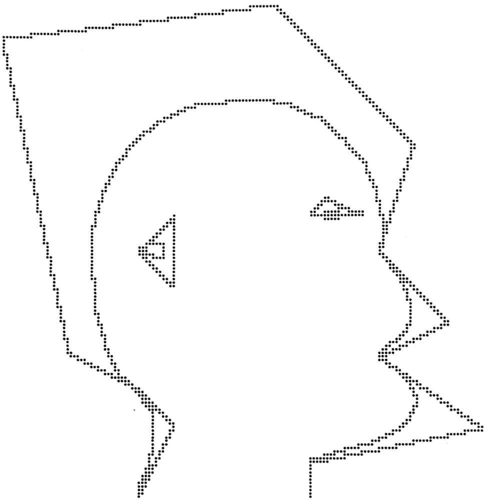


Figure 6.11. Curve fitted mechanically.

The method is related to using the curved stencils employed by artists that seem to have different names around the world. In England and North America they are called "French curves" (Figure 6.12). Bezier fitting, as the process is called, uses the sets of curves that Figures 6.13A and 6.13B illustrate, depending on the number of points in the angular outline that you want to fit. These are drawn by the program BEZFUN. It is given separately, as a preliminary to the program BEZFIT, which does the actual fitting, because the Bezier curves form interesting patterns, and the brevity of the program helps clarify the mathematical notation usually employed to describe it.

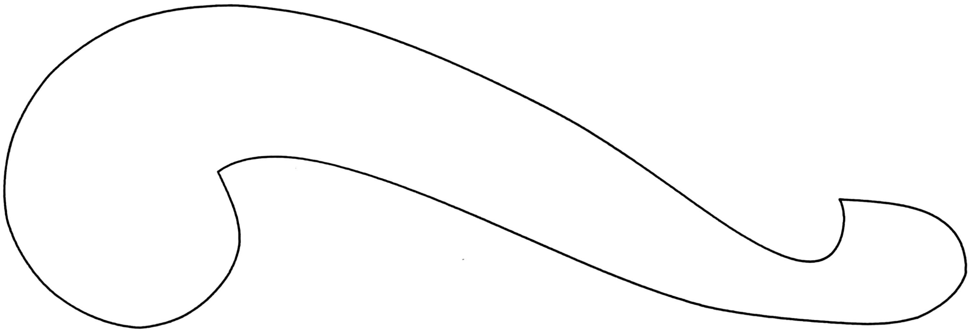


Figure 6.12. Outline of French curves.

The Bezier functions are plotted on axes that go from 0 to 1. The variable on the horizontal axis often is written as u, instead of x, in descriptions of these functions. Each function is identified by n, one less than the total number of points in the shape to be fitted, and i, that takes values $0-n$. For just two points, n is 1, and the functions with $i=0$ and $i=1$ are just straight lines:

$$b(1,0,u)=1-u \qquad \text{and} \qquad b(1,1,u)=u$$

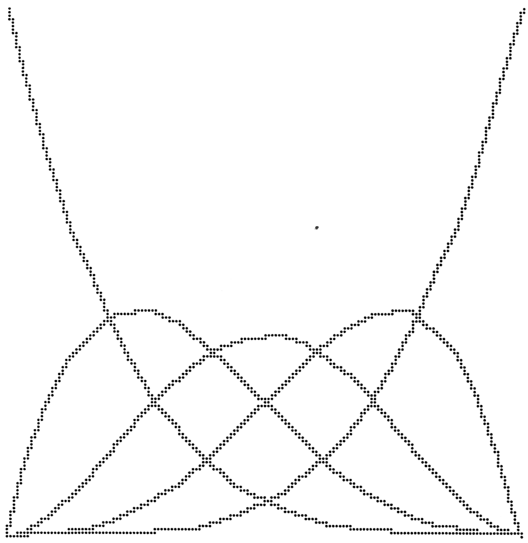


Figure 6.13A. N=4.

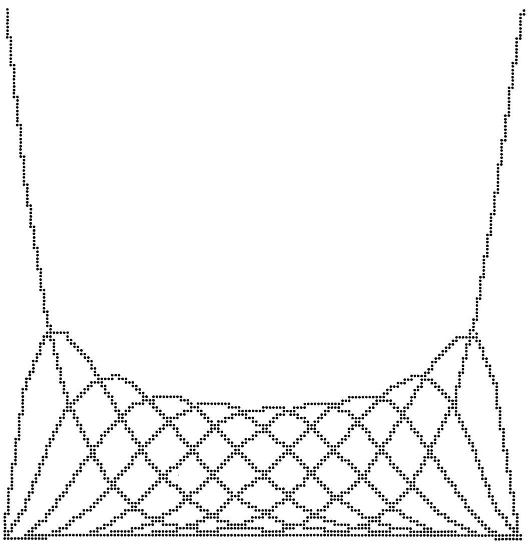


Figure 6.13B. N=10.

We use the notation $b(n,i,u)$ for a Bezier function. When n is 2, the three Bezier functions are

$$b(2,0,u)=(1-u)^2, \qquad b(2,1,u)=2u(1-u), \qquad b(2,2,u)=u^2$$

and when n is 3, the four Bezier functions are

$$b(3,0,u)=(1\text{-}u)^3, \qquad b(3,1,u)=3u(1\text{-}u)^2$$
$$b(3,2,u)=3u^2(1\text{-}u), \qquad b(3,3,u)=u^3$$

The way these formulas can be extended to higher values of n is obvious to readers who know the binomial theorem for expanding $(1+x)^n$. For the mathematically inclined and/or purposes of gamesmanship, these are just the binomial probability distribution functions. Which goes to show how simple a thing can be covered with a four-word, 15-syllable, 40-letter name!

The general expression for a Bezier function can be written out, in factorial notation, and a program written to compute each function separately. It is faster, however, to start with

$$b(n,0,u)=(1\text{-}u)^n$$

and to use the formula

$$b(n,i,u)=\frac{(n-i+1)}{i}\times\frac{u}{(1-u)}\times b(n,i-1,u)$$

Setting i to 1 computes $b(n,1,u)$ from $b(n,0,u)$. Setting i to 2 computes $b(n,2,u)$ from $b(n,1,u)$. And so on, to $i=n$. The calculation is shortened when u is 0 or 1 by the fact that $b(n,i,0)$ is zero unless $i=0$, and $b(n,i,1)$ is zero unless $i=n$.

The main variables in the program are

`N`	The number of control points, that is $n+1$
`I,U`	Correspond to i, u

NU	Number of points for which the functions are plotted including the end points of the hull
NP	The number of points excluding the end points
IP	Takes values 0, 1, ..., NP for these successive points; that is, U=0 when IP=0 and U=1 when IP=NP
BN(I)	Value of $b(n,i,u)$ for current value of u
U0	Previous value of u, along the curves
B0(I)	Value of $b(n,i,u)$ for previous value of u, along the curves

Suggested variations

1. Fill the interstices in the pattern formed by a set of Bezier functions.
2. Use these patterns for some animation effects.

The BEZFUN program

```
100  REM *** BEZFUN-PLOTS BEZIER FUNCTIONS ***
110  TEXT : HOME : HGR: HCOLOR=3: POKE 34,20
120  HOME : INPUT "NUMBER OF CONTROL POINTS: ";NC
130  N=NC-1: IF N<2 THEN 120
140  INPUT "NUMBER OF POINTS PER CURVE: ";NU
150  NP=NU-2: IF NP<1 THEN 120
170  DIM B0(N),BN(N)
180  B0(0)=1
190  FOR IP=1 TO NP: REM *** CYCLE THROUGH THE POINTS ***
200  U=IP/(NP+1):BN(0)=(1-U) ^ N
220  BN(I)=(N-I+1)/I*U/(1-U)*BN(I-1)
250  HPLOT 150*U0,159-150*B0(I) TO 150*U,159-150*BN(I): REM *** JOIN POINTS ***
260  B0(I)=BN(I)
270  NEXT I
280  U0=U
290  NEXT IP
300  FOR I=0 TO N-1: REM *** COMPLETE THE CURVES THAT END AT 1 ***
310  HPLOT 150*U0,159-150*BN(I) TO 150,159
320  NEXT I
330  HPLOT 150*U0,159-150*BN(N) TO 150,9: REM *** THIS ENDS AT 0 ***
340  END
```

6.5 DOING THE ACTUAL SMOOTHING

Now we can use the "smoothing curves" that have just been described. The program BEZFIT displays the main menu:

```
TYPE 1 TO PLOT CURVE,  2 TO ERASE CURVE,
     3 TO DRAW LINE,    4 TO CLEAR SCREEN,
     5 TO ALTER CURVE, 6 TO QUIT:
```

If you type 1, the program prompts for the number of points that determine the shape you want to fit. These are called the "control points." The sequence of straight lines that join the control points form the "hull." This must be convex, that is, it cannot go in and out, but it can be closed.

The program gives the user the option of displaying the control points, and then prompts for their coordinates. After each pair of coordinates has been typed, the program allows these to be kept or changed. The smooth curve that is drawn after all the control points have been specified consists of a succession of short lines that join a series of intermediate points that the program computes. It prompts for the number of these to be included between the ends, and computes their coordinates by means of the Bezier process described later in this section. The program then gives the user the choice of seeing just the curve or the hull as well, and plots one or both accordingly.

The data

number of control points=4,

coordinates (0,0), (40,80), (240,80), (279,0)

produce the display shown in Figure 6.14. The data

number of control points=5,

coordinates (0,0), (120,0), (120,120), (0,120), (0,0)

produce the display shown in Figure 6.15.

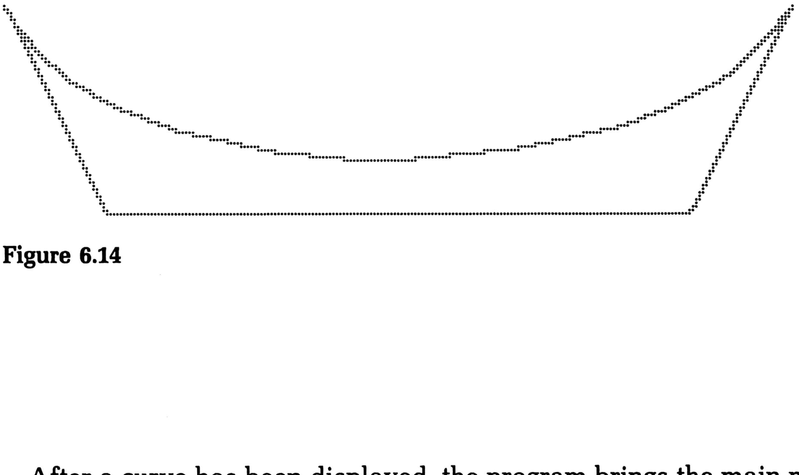


Figure 6.14

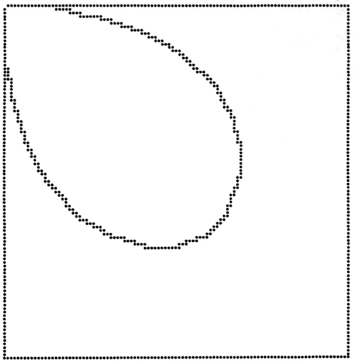


Figure 6.15

After a curve has been displayed, the program brings the main menu back to the text area of the screen. The user can specify the hull of another curve to be plotted or take one of the other actions. You type 2 to erase a curve, by then repeating the input that was used to draw it. If you type 3, the program prompts for the end points of a straight line and, after you have typed their coordinates, draws it. Typing 4 clears the screen.

If you type 5, the secondary menu is displayed:

```
TYPE 1 (ERASE LAST CURVE), 2 (DISPLAY
COORDINATES), 3 (ALTER A POINT),
4 (DISPLAY NEW CURVE), 5 (RETURN TO
MAIN MENU):
```

At this point, typing 1 erases the curve that was just drawn, and typing 2 displays the coordinates of its control points. These are retained in memory after the curve has been erased, and until another one is started. Typing 4 allows you to specify the sequence number of a point in the last curve. Its coordinates are displayed and the program prompts for new values. You can alter several points in this way, and then request the new curve to be drawn by typing 4. The program thus can be used to adjust a curve interactively.

The Bezier process works as follows. The variable u featured in the preceding section takes values from 0 to 1 as we move along the curve that is being constructed. If you type 9 for the number of points, u takes the values 0, 0.1, 0.2, ..., 1.0. The x coordinate of each of the points that is plotted for a particular value of u is found from the formula

$$x = x_0 b(n,0,u) + x_1 b(n,1,u) + \ldots + x_n b(n,n,u)$$

where x_0, x_1,...x_n are the x coordinates of the control points. The b are just the Bezier functions that were described in the preceding section for the current value of u. The y coordinate is found from

$$y = y_0 b(n,0,u) + y_1 b(n,1,u) + \ldots + y_n b(n,n,u)$$

where y_0, y_1, ...,y_n are the y coordinates of the control points.

These formulas produce a smooth curve inside the hull formed by joining the control points because of the shapes of the Bezier functions. For $n=3$, these are as shown in Figure 6.16. The process starts with $u=0$. Because $b(3,0,u)$ is 1, and all the other b are 0, the x, y coordinates of the point that is plotted are simply x_0, y_0, the coordinates of the first control point. As u increases, the weights given to x_0 and y_0 in the formulas for x and y decrease, but the weights given to the coordinates of the other control points increase. This is most rapid for x_1, y_1 at first, so that by the time u corresponds to the maximum in $b(3,1,u)$, the coordinates of this control point are dominant. Then the influence of this point starts to diminish, as $b(3,1,u)$ begins to decrease, but the influence of x_2, y_2 builds up as $b(3,2,u)$ reaches its maximum. Then, in the last lap, $b(3,3,u)$ builds up to 1, while the other b drop to zero. The coordinates x_3, y_3 of the final control point become increasingly important in determining the coordinates x, y of the points that are plotted until, when $u=1$, all the b except the last one are zero, and the curve ends at x_3, y_3, on the final control point itself.

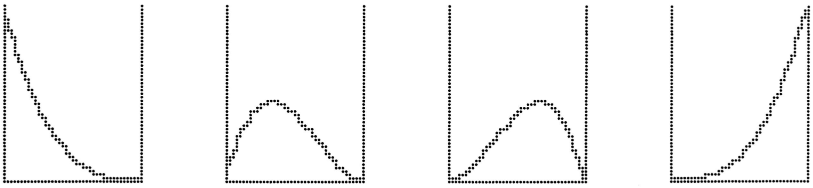


Figure 6.16

The main variables in the program are

`NC`	The number of control points
`N`	The number of control points minus 1

`I`	Takes values 0, 1, ..., N for successive control points
`X(I)`	*x* coordinate of the Ith control point
`Y(I)`	*y* coordinate of the Ith control point
`NP`	The number of points to be plotted between the end points
`IP`	Takes values 0, 1, ..., NP+1 for the points that are plotted (IP=0 and NP+1 for the two end points)
`U`	The parameter that goes from 0 to 1 along the curve
`XP(IP)`	The *x* coordinate of the IPth point to be plotted
`YP(IP)`	The *y* coordinate of the IPth point to be plotted
`B(I)`	The Bezier function $b(n,i,u)$ for the current values of N, I, U

The program prompts for the input data, and when the user has defined a hull, it goes through a FOR loop for successive values of IP. The Bezier functions are computed for the current value of U, using another FOR loop; the *x*, *y* coordinates of the points to be plotted, that is, XP(IP),YP(IP), are computed in a third loop. These values are saved until IP has reached its limit. The program then draws the hull, if requested, and the smooth curve. It then restarts with a prompt for more data.

Suggested variations

1. Explore the effects of increasing the number of control points and using several control points that are very close together or actually coincident.
2. Reduce the amount of storage by keeping just the coordinates of the previous point on the smooth curve, while those of the current point are being computed, and joining these in the loop controlled by IP.
3. Allow the coordinates of the smooth curves to be saved in text files and read and displayed in later runs that bypass the smoothing process.
4. Provide axes along the edges of the screen.
5. Specify the position of the control points by a cursor. (See, for example, Section 3.3 for a way to program this.)
6. If you have a paddle or joystick control on your Apple, use this to control the positions of the points.

The BEZFIT program

```
100  REM   *** BEZFIT FOR INTERACTIVE BEZIER FITTING ***
110  DIM X(20),Y(20),B(20),XP(100),YP(100)
120  HGR: HCOLOR=3: POKE 34,20: HOME :
130  HOME : PRINT "TYPE 1 TO PLOT CURVE, 2 TO ERASE CURVE,"
140  PRINT " 3 TO DRAW LINE, 4 TO CLEAR SCREEN,"
150  INPUT " 5 TO ALTER A CURVE, 6 TO QUIT: ";OP%
160  HOME : ON OP% GOTO 200,410,430,460,510,730
170  PRINT "INVALID REQUEST: ";OP%
180  FOR DE=1 TO 1000: NEXT DE: GOTO 130
190  REM *** DEFINE A HULL ***
200  HOME : INPUT "NUMBER OF CONTROL POINTS: ";NC
210  N=NC-1
220  HOME : INPUT "DISPLAY POINTS?(Y OR N) ";Q$:Q$=LEFT$(Q$,1)
230  FOR I=0 TO N: REM *** INPUT CYCLE FOR THE CONTROL POINTS ***
240  HOME : PRINT "COORDINATES OF CONTROL POINT ";I+1;: INPUT ": ";X,Y
```

```
 250  IF X<=279 AND Y<=159 AND X>=0 AND Y>=0 THEN 270
 260  PRINT "***ILLEGAL VALUE***": GET A$: GOTO 240
 270  X(I)=X:Y(I)=Y
 280  IF Q$="N" THEN 300:REM *** BRANCH IF NOT TO PLOT ***
 290  HPLOT X,Y
 300  HOME : INPUT "ADEQUATE?(Y OR N) ";RQ$
 310  IF RQ$<>"N" THEN 330
 320  HCOLOR=0: HPLOT X,Y: HCOLOR=3: GOTO 240: REM *** IT WAS ERASED ***
 330  NEXT I: REM *** END OF INPUT CYCLE ***
 340  HOME : INPUT "NO. OF POINTS TO BE PLOTTED (<=50): ";NU
 350  NP=NU-2: IF NP>48 THEN 360
 360  IF Q$<>"Y" THEN 390
 370  HCOLOR=0: REM *** ERASE THE CONTROL POINTS ***
 380  FOR I=0 TO N: HPLOT X(I),Y(I): NEXT I
 390  HCOLOR=3: GOSUB 1010: GOTO 130
 400  REM *** ERASE A CURVE ***
 410  HCOLOR=0: GOTO 200
 420  REM *** DRAW A STRAIGHT LINE ***
 430  INPUT "END POINTS OF THE LINE? ";X1,Y1,X2,Y2
 440  HPLOT X1,Y1 TO X2,Y2: GOTO 130
 450  REM *** CLEAR THE SCREEN ***
 460  HGR: HCOLOR=3: GOTO 130
 500  REM *** ALTER A CURVE ***
 510  HOME : PRINT "TYPE 1 (ERASE LAST CURVE), 2 (DISPLAY"
 520  PRINT "COORDINATES), 3 (ALTER A POINT),"
 530  PRINT "4 (DISPLAY NEW CURVE), 5 (RETURN TO "
 540  INPUT "MAIN MENU): ";OQ%
 550  HOME : ON OQ% GOTO 590,610,690,390,130
 560  PRINT "INVALID REQUEST: ";OQ%
 570  PRINT "PRESS ANY KEY TO TRY AGAIN": GET A$: GOTO 510
 580  REM *** ERASE THE LAST CURVE **
 590  HCOLOR=0: GOSUB 1010: HCOLOR=3: GOTO 510
 600  REM *** DISPLAY THE COORDINATES ***
 610  TEXT: HOME: INVERSE: PRINT "I.D.#","X-COORD","Y-COORD": NORMAL: VTAB 4
 620  FOR I=0 TO N
 630  PRINT I+1,X(I),Y(I)
 640  NEXT I
 650  PRINT "HIT ANY KEY TO CONTINUE ";: GET A$
 660  POKE -16304,0: POKE -16297,0: POKE 34,20: HOME
 670  GOTO 510
 680  REM *** ALTER A CONTROL POINT ***
 690  INPUT "ID# OF POINT: ";I:I=I-1
 700  PRINT "OLD COORDS: ";X(I),Y(I)
 710  INPUT "NEW COORDS: ";X(I),Y(I)
 720  GOTO 510
 730  TEXT : HOME : END
1000  REM *** COMPUTE BEZIER FUNCTIONS AND PLOT THE CURVE ***
1010  XP(0)=X(0):YP(0)=Y(0)
1020  IF OP%=2 THEN  HCOLOR=0
```

```
1030  FOR IP=1 TO NP: REM *** CYCLE FOR SUCCESSIVE PLOTTING POINTS ***
1040  U=IP/(NP+1):B(0)=(1-U) ^ N
1050  FOR I=1 TO N: REM *** CYCLE FOR SUCCESSIVE ORDERS ***
1060  B(I)=(N-I+1)/I*U/(1-U)*B(I-1)
1070  NEXT I: REM *** END CYCLE FOR SUCCESSIVE ORDERS ***
1080  XP(IP)=0:YP(IP)=0
1090  FOR I=0 TO N: REM *** COMPUTE THE WEIGHTED COORDINATES ***
1100  XP(IP)=XP(IP)+B(I)*X(I)
1110  YP(IP)=YP(IP)+B(I)*Y(I)
1120  NEXT I: REM *** END OF COMPUTATION OF COORDINATES ***
1130  NEXT IP: REM *** END OF CYCLE FOR SUCCESSIVE PLOTTING POINTS ***
1140  XP(NP+1)=X(N):YP(NP+1)=Y(N)
1150  HOME : INPUT "TYPE (C)URVE OR (B)OTH CURVE AND HULL: ";IN$
1160  IF IN$="C" THEN 1200
1170  FOR I=1 TO N: REM *** DRAW THE HULL ***
1180  HPLOT X(I),Y(I) TO X(I-1),Y(I-1)
1190  NEXT I
1200  FOR IP=1 TO NP+1: REM *** DRAW THE CURVE ***
1210  HPLOT XP(IP),YP(IP) TO XP(IP-1),YP(IP-1)
1220  NEXT IP
1230  HCOLOR=3
1240  RETURN
```

6.6 GOING INTO ORBIT

You can get an animated view of satellites in orbit, or electrons going round a nucleus, or anything going around anything else in paths that are circles or ellipses, by using the program ORBIT. This puts the sun, nucleus, or whatever it is that the satellites, electrons, and so forth go around at the center of the screen. The objects are shown as small squares that spin as they rotate (Figure 6.17). The program prompts for the number of orbiting objects, and their speed and size, as well as two numbers that define the orbit of each. When the orbit is circular, these numbers are both equal to the orbit's radius. When the orbit is an ellipse, the numbers are the lengths of its horizontal and vertical semiaxes. An interesting display is obtained by specifying 5 objects with speed, size, and axis values in the ranges 2–12, 1–8, and 10–120 and 10–70, respectively.

The program uses a coordinate system with its origin at the center of the screen. It computes the coordinates of an orbiting object from

$$x = a\cos\theta, \qquad y = b\sin\theta$$

where a and b are the input numbers that define the orbit, and the angle θ is increased, by an amount proportional to the speed of the object, from one cycle of the animation to the next. This is approximately correct for planets going around the sun and for moons going around a planet with orbits that are only slightly elliptical. For greater accuracy, however, and for comets, the sun should be at a focus of the ellipse. (Definitions of terms may be found in books on geometry.)

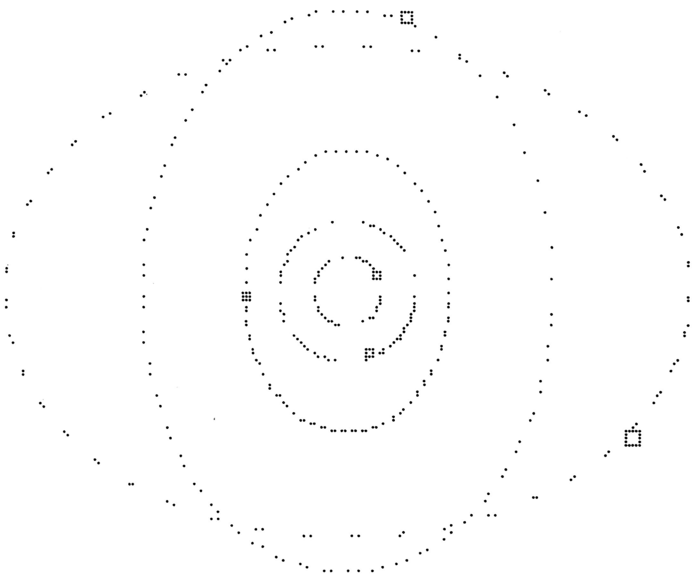


Figure 6.17

The main variables in the program are

`TV(2,314)`	Array that contains values of sines and cosines used to compute the positions of the orbiting objects
`NO`	Number of orbiting objects
`L`	Takes values 1, 2, ..., NO for these
`XF(L)`	Length of horizontal semiaxis of Lth object
`YF(L)`	Length of vertical semiaxis of Lth object
`XN(L)`	x coordinate of Lth object
`YN(L)`	y coordinate of Lth object
`XO(L)`	Previous value of XN(L)
`YO(L)`	Previous value of YN(L)
`PN(L)`	Angle of rotation of Lth object about center
`RN(L)`	Angle of spin of Lth object (in units of $45\pi/8$ radians)
`RO(L)`	Previous value of RN(L)

The program first stores the codes for a small square as a shape table and computes a table of sines and cosines to make the animation proceed quickly. It prompts for the number of orbiting objects and their properties, and then executes a cycle that orbits endlessly.

This main cycle contains an inner cycle that deals with each orbiting object in turn. In this inner cycle PN(L) is increased by the angular speed SP(L), to give the new angular position of the Lth object in units that correspond to successive elements of TV. The angle of spin RN(L) also is increased by the angular velocity SP(L). The current coordinates are computed from the formulas given earlier. The object is drawn at its new position and orientation by use of the ROT and DRAW commands, and erased from its previous position.

Suggested variations

1. Allow the elliptical orbits to have axes that are tilted.
2. Use different shapes for the objects.

3. Use the equations that plot elliptical orbits with the sun at a focus instead of at the center of the ellipse.
4. Speed up the calculation of sines and cosines at the beginning of the program. (*Hint:* Compute the sine and cosine of 1° and then use the formulas for the sine and cosine of the sum of two angles to find these for 2°, 3°, and so on.)

The ORBIT program

```
100  REM *** ORBIT-PLOTS ELLIPTICAL PATHS ***
110  LOMEM: 16384
120  DIM TV(2,314)
130  HGR: HCOLOR=3: POKE 34,20: HOME : PRINT "*** PLEASE WAIT-COMPUTING ***"
140  SCALE=7: ROT=0:P=0
150  FOR I=7676 TO 7682: READ V: POKE I,V: NEXT I: REM *** STORE SHAPE TABLE ***
160  POKE 232,252: POKE 233,29: REM *** STORE ITS BASE ADDRESS ***
170  DATA 1,0,4,0,44,62,0
180  FOR A=0 TO 6.28 STEP .02
190  TV(1,P)=COS(A):TV(2,P)=SIN(A):P=P+1
200  NEXT A
210  HOME : INPUT "HOW MANY ORBITING OBJECTS? ";NO
220  IF NO>10 THEN  PRINT "NO MORE THAN 10":NO=10
230  FOR L=1 TO NO
235  HOME : PRINT "OBJECT ";L;" OF ";NO
240  INPUT "HALF LENGTHS OF AXES=";XF(L),YF(L)
250  INPUT "SPEED AND SIZE=";SP(L),SC(L)
260  NEXT L
270  FOR L=1 TO NO
280  RN(L)=RN(L)+SP(L)
290  IF RN(L)>64 THEN RN(L)=RN(L)-64
300  PN(L)=PO(L)+SP(L)
310  IF PN(L)>314 THEN PN(L)=PN(L)-314
320  NEXT L
330  FOR L=1 TO NO
340  XN(L)=140+XF(L)*TV(1,PN(L))
350  YN(L)=80+YF(L)*TV(2,PN(L))
360  NEXT L
370  FOR L=1 TO NO
380  SCALE=SC(L): ROT=RO(L)
390  XDRAW 1 AT XO(L),YO(L)
400  ROT=RN(L)
410  XDRAW 1 AT XN(L),YN(L)
420  HPLOT XO(L),YO(L)
430  NEXT L
440  FOR L=1 TO NO
450  XO(L)=XN(L):YO(L)=YN(L)
460  RO(L)=RN(L):PO(L)=PN(L)
470  NEXT L
480  GOTO 270
```

6.7 A LITTLE ROULETTE

You can display dozens of interesting curves with the program ROULETTE. Some examples are shown in Figures 6.18A–6.18C and 6.19A–C. These include the astroid, cardioid, and nephroid that you have seen already. The program begins by prompting for the letter E or H, to select the general style of the first or second of these sets of curves, and then for the number of cusps and the size of the curve. For the curves just shown, the input is

E,2,5 E,3,5 E,4,5

H,3,5 H,4,5 H,5,5

The size should be in the range 2–8. The letters E and H stand for epicycloid and hypocycloid, so called because these two kinds of curve are drawn by a pen on the rim of a circle that is rolled around the outside or the inside of a larger circle. The curves also are called roulettes because they can be drawn this way. The ancients believed that the heavenly bodies moved along paths with these shapes.

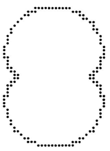


Figure 6.18A

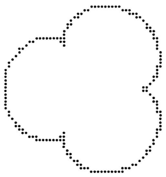


Figure 6.18B

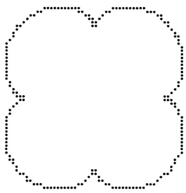


Figure 6.18C

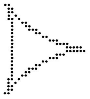


Figure 6.19A

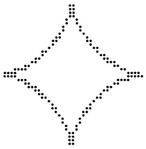


Figure 6.19B

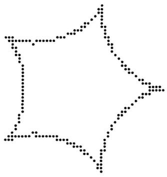


Figure 6.19C

The program is very short. It calculates the x, y coordinates of the points on an epicycloid, relative to its center, from

$$x = na\cos t - a\cos nt$$

$$y = na\sin t - a\sin nt$$

where n is the number of cusps plus 1, a determines the size, and t is an angle that goes from 0 to 2π. These define a cardioid when n is 2, and a nephroid when

n is 3. This kind of definition of a curve, which uses separate equations for x and y in terms of a third variable (in the present case t), is called a "parametric definition," and the third variable is called the "parameter."

The corresponding equations for a hypocycloid are

$$x = nr \cos t + a \cos nt$$

$$y = nr \sin t + a \sin nt$$

where n is the number of cusps minus 1. These define a deltoid when n is 2 and an astroid when n is 3.

The main variables in the program are N, A, and T, corresponding to n, a, and t in these formulas. The program prompts for data, and goes through a cycle for successive values of t, plotting the points with coordinates computed from the appropriate formulas.

Suggested variations

1. See what happens when you type the ratio of two integers in response to the prompt for the number of cusps.
2. Draw cardioids with cusps at the left, top, and bottom. (*Hint:* Reverse the sign of the expression for x to put the cusp at the left; interchange the expressions for x and y to put the cusp at the top.)
3. Draw a nest of concentric cardioids. (*Hint:* Use different values of r.)
4. Draw cardioids using the polar equation that relates the values of r and θ in Figure 6.20. The equation is

$$r = 2a(1 - \cos \theta)$$

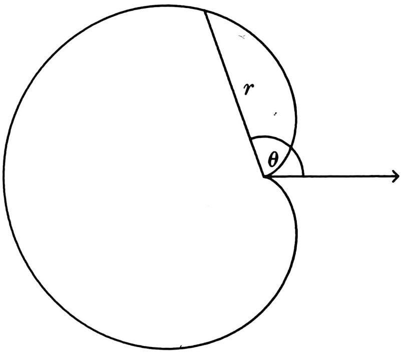


Figure 6.20

5. Draw a nest of cardioids with their cusps touching. (*Hint:* Just use the polar equation, with different values of a.)
6. Draw two cardioids touching "face to face" (Figure 6.21). [*Hint:* Left-hand cardioid is given by $r_1 = 2a(1 - \cos \theta_1)$ and right-hand cardioid is given by $r_2 = 2a(1 - \cos\theta_2)$, where r_1 is measured from $(-a/2,0)$ and r_2 from $(+a/2,0)$, θ_1 is measured from the positive direction of the x axis, and θ_2 is measured from the negative direction of the x axis.]

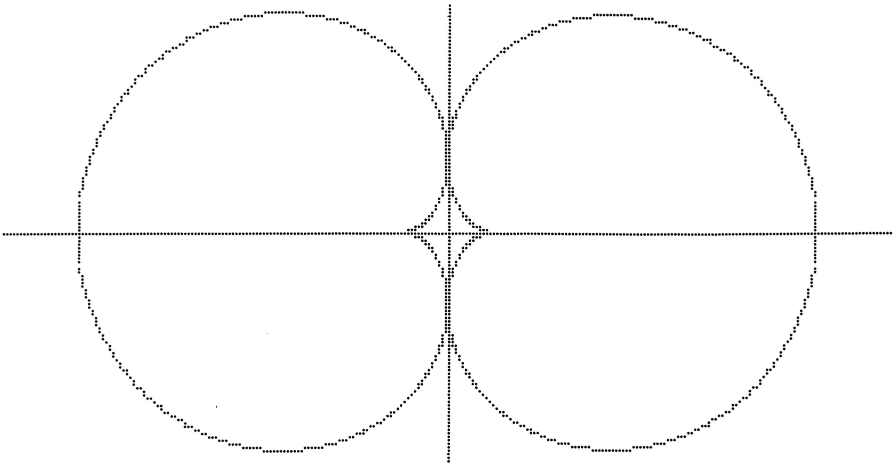


Figure 6.21

7. Draw a nest of nephroids (Figure 6.22). [*Hint:* The outermost is given by

   $$x = a(3\cos t - \cos 3t), \qquad y = a(3\sin t - \sin 3t)$$

   the next by

   $$x = (a/2)(3\cos t + \cos 3t), y = (a/2)(3\sin t + \sin 3t)$$

   the next by

   $$x = (a/4)(3\cos t - \cos 3t), y = (a/4)(3\sin t - \sin 3t)$$

   and so on.]

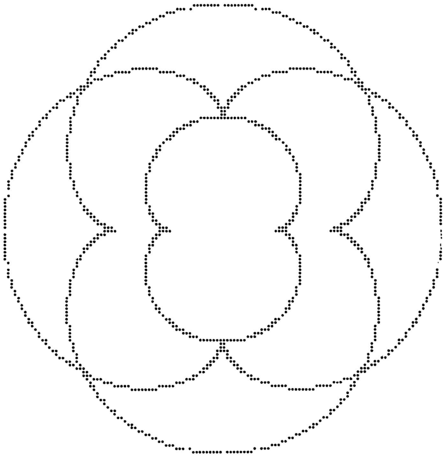


Figure 6.22

The ROULETTE program

```
100  REM *** ROULETTE-DRAWS CYCLOIDS ***
110  TEXT : HOME : HGR: HCOLOR=3
120  HOME : VTAB 21
```

```
130  PRINT "TYPE E FOR EPICYCLOID,"
140  PRINT " H FOR HYPOCYCLOID,"
150  INPUT " Q TO QUIT: ";A$
160  HOME : VTAB 21
170  IF A$="E" THEN 220
180  IF A$="H" THEN 310
190  IF A$="Q" THEN END
200  GOTO 120
210  REM *** EPICYCLOID ***
220  INPUT "CUSPS,SIZE: ";N,A
230  N=N+1
240  IF N*A>80 THEN A=N/80
250  FOR T=0 TO 6.28 STEP .02
260  X=140+N*A*COS(T)-A*COS(N*T)
270  Y=80+N*A*SIN(T)-A*SIN(N*T)
280  HPLOT X,Y
290  NEXT T
300  GOTO 130
310  REM *** HYPOCYCLOID ***
320  INPUT "CUSPS,SIZE: ";N,A
330  N=N-1
340  FOR T=0 TO 6.28 STEP .02
350  X=140+N*A*COS(T)+A*COS(N*T)
360  Y=80+N*A*SIN(T)-A*SIN(N*T)
370  HPLOT X,Y
380  NEXT T
390  GOTO 120
```

6.8 HOW TO MAKE WAVES

The sine wave is one of the commonest patterns seen on the screen of electronic test gear (Figure 6.23). It just goes on and on, repeating itself for as far as you want, in both directions.

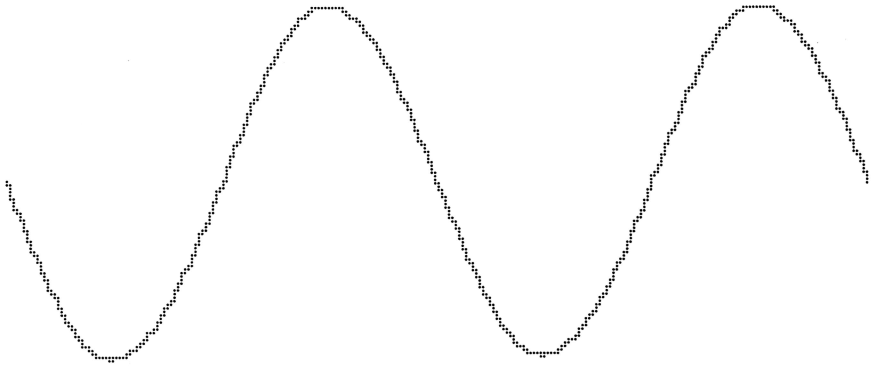


Figure 6.23

The program SINEWAVE begins by prompting

```
TYPE 1 FOR SAMPLE SINE WAVES,
     2 TO SPECIFY THE PARAMETERS,
     3 TO QUIT:
```

If you type 1, the program displays Figure 6.24 on the graphics screen. The end points of the axes are labeled, and the equation is displayed at the top of the screen. A secondary menu appears in the text area:

```
TYPE 1 FOR ANOTHER SAMPLE SINE WAVE,
     2 TO INPUT YOUR OWN PARAMTERS,
     3 TO QUIT:
```

If you type 1, the display changes to Figure 6.25, and the secondary menu reappears in the text area.

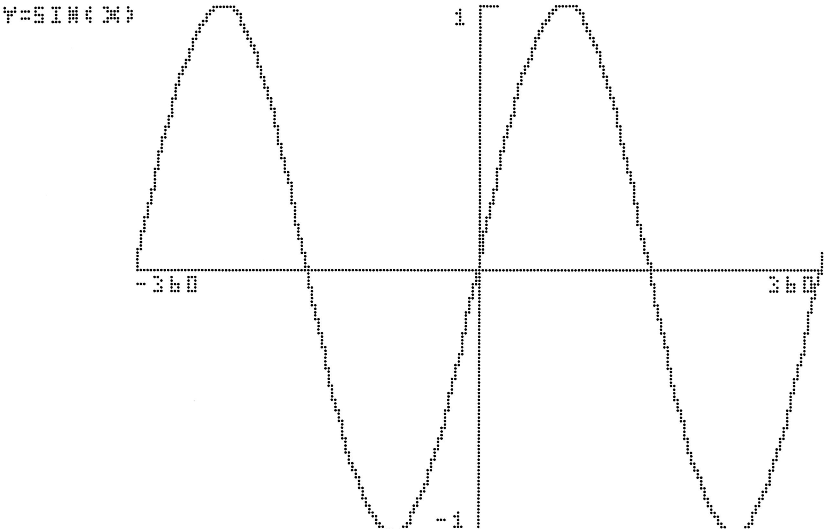


Figure 6.24

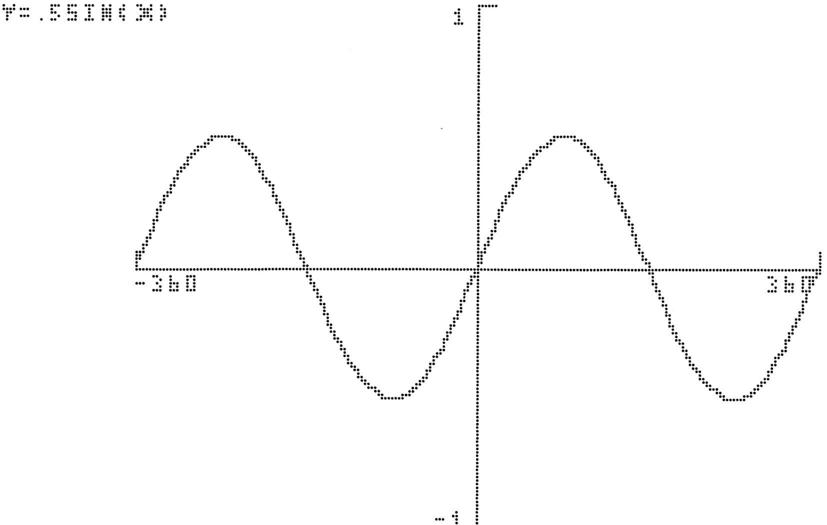


Figure 6.25

If you continue to type 1 after each display has appeared, you get, in succession, Figures 6.26A and 6.26B.

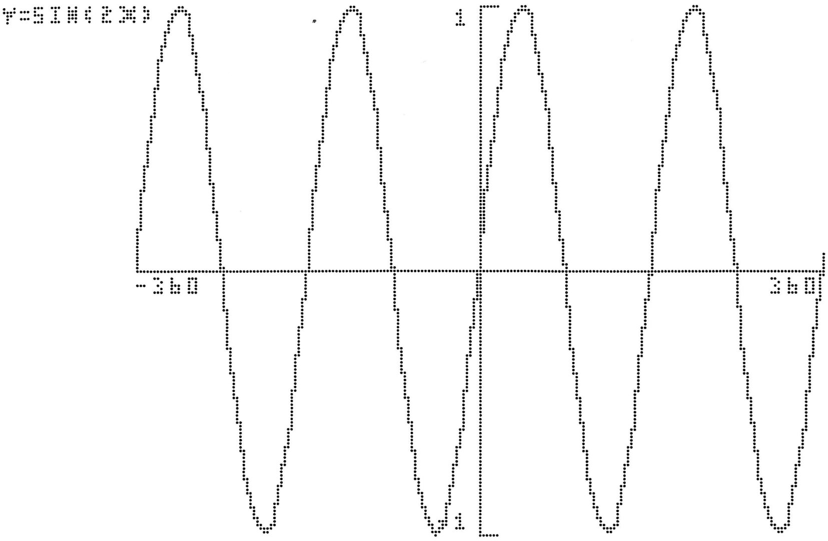


Figure 6.26A

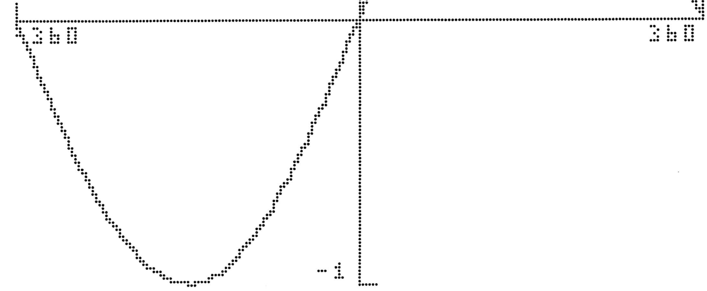


Figure 6.26B

For both of these, x goes from -2π to 2π, and y from -1 to 1. The amplitude, that is, the distance from the peaks and troughs to the horizontal median (in this case the x axis), is 1 for both, but the frequency is 2 for the wave that makes two complete cycles between -2π and 2π. It is 0.5 for the wave that makes only half a complete cycle between 0 and 2π.

To see the effect of changing the x shift and y shift, continue to type 1 after each display has appeared. You get Figures 6.27A and 6.27B. For both of these curves, the x axis still goes from -2π to 2π and the y axis from -1 to 1. The frequency is 1 for both. The amplitude is 1, the x shift is $\pi/2$, and the y shift is 0 for the first. The amplitude is 0.5, the x shift is 0, and the y shift is 0.5 for the second. Thus the x shift moves the wave horizontally, and the y shift moves it vertically.

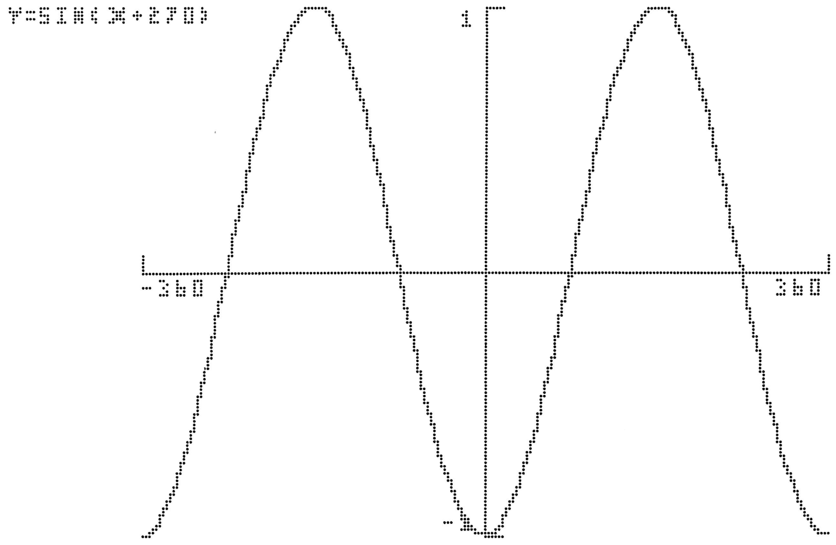


Figure 6.27A

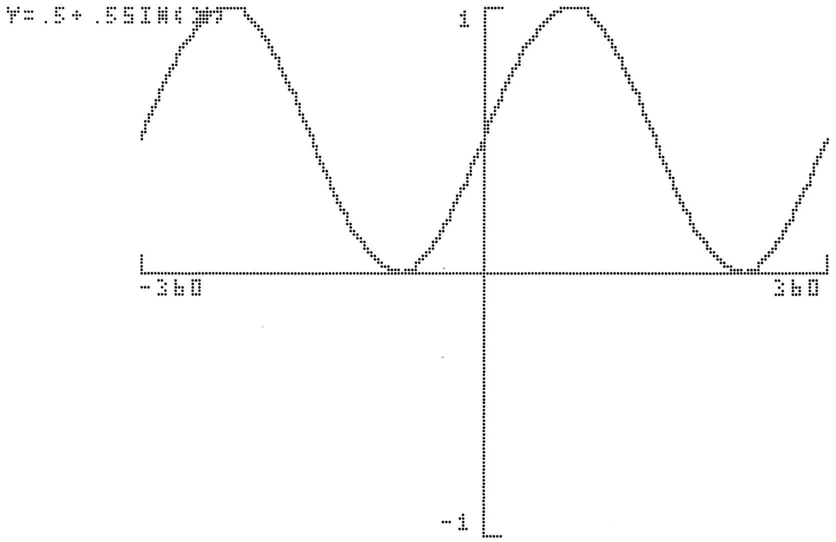


Figure 6.27B

You now can type 2 to specify your own parameters. The program prompts in turn for the amplitude, frequency, x shift and y shift, and minimum and maximum values of x and y. For each of these, it displays the previous values and requests the new values. When all of these have been specified, the program draws the sine wave on the screen, and displays the equation at the top.

It also displays another self-explanatory menu:

```
TYPE 1 TO INPUT A NEW SET OF PARAMETERS,
     2 FOR A PRESET SAMPLE,
     3 TO QUIT:
```

You can continue plotting sine waves with parameters that were preset or that you specify.

The program simply plots points with x, y coordinates measured from the center of the screen. The x values are stepped through the appropriate range by a FOR loop. The y values are computed by

$$y = v + a \sin (fx + h)$$

where v is the y shift, a is the amplitude, f is the frequency, and h is the x shift.

The main variables in the program are as follows

`A`	The amplitude of a sine wave
`F`	Its frequency
`XS,YS`	Its x and y shifts
`XV,XH`	The limits on the x axis on which it is plotted
`YV,YH`	The limits on the y axis
`IM`	The number of preset sine waves
`W,H`	The width and height of the characters that are drawn on the screen
`XI`	The number of points that are used to plot a sine wave
`CF`	The factor to convert from degrees to radians
`OP`	The option that is selected from the main menu
`IQ`	The index of the preset sine wave under current attention
`XL,YL`	The screen coordinates for the start of a string
`ST$`	The string to be displayed
`XA,YA`	The x coordinate of the y axis and the y coordinate of the x axis, respectively
`SX,SY`	The factors to convert from user to screen coordinates
`XU,YU`	The user coordinates of a point to be plotted on a sine wave
`XN,YN`	The corresponding screen coordinates
`XO,YO`	The screen coordinates of the previous point
`OP`	The option selected from a secondary menu
`I`	The index of the successive characters in a string that is displayed
`CH$`	The character under current attention in this string

The operation of the program is straightforward. The parameters for the preset sine waves are read from DATA statements. When the user types the parameters, the program displays the previous values as part of the prompt and then displays the new values. The two branches coalesce to display the equation. The subroutines to draw a string are the same as in POLYPLOT and POLYFIT. Similar tactics are used to suppress the characters 0, 1, and + when these are redundant, to draw axes, and to draw and label the tick marks.

Suggested variations

1. Let the user specify the plotting interval.
2. After a sine wave has been plotted, let the user control whether the screen is cleared or the next sine wave is superimposed, using the same ranges of x and y.
3. Draw the sine waves to go down the screen instead of across it. That is, make x a sine function of y.
4. Draw sine waves with amplitude 20 units or less (a) along the top and bottom of the screen, (b) along the left and right edges, (c) along all four edges, forming a border, and (d) along lines parallel to the edges of the screen, forming concentric patterns (Figure 6.28). (e) Allow different amplitudes and frequencies in the sine waves that follow the different rectangles.

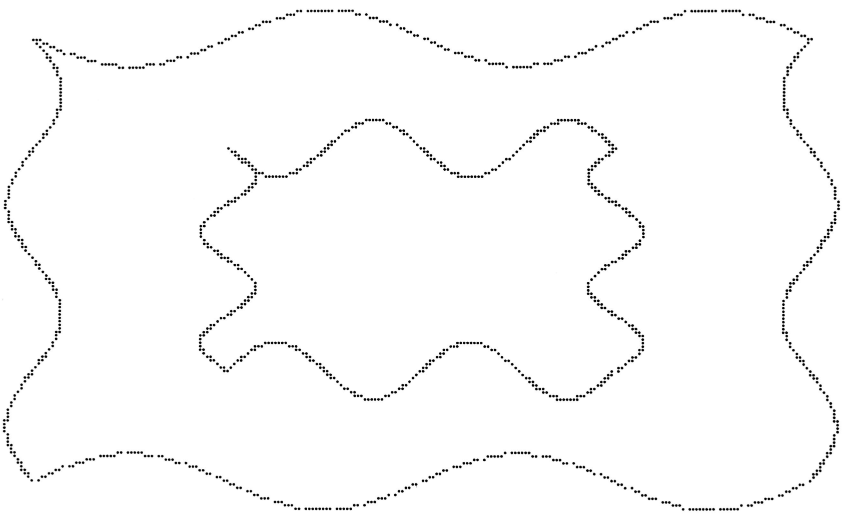


Figure 6.28

5. Draw sets of parallel sine waves, for example, Figure 6.29.

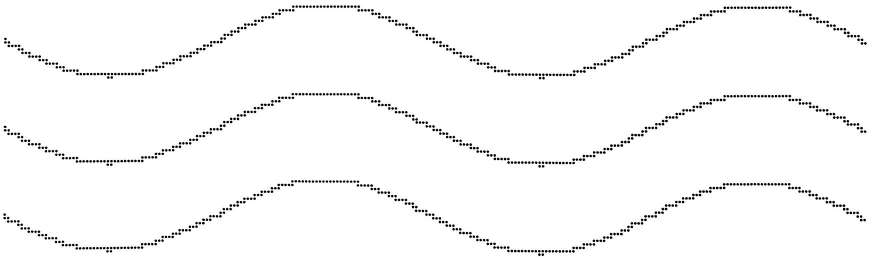


Figure 6.29

6. Shade the region between pairs of such waves, for example, Figure 6.30.

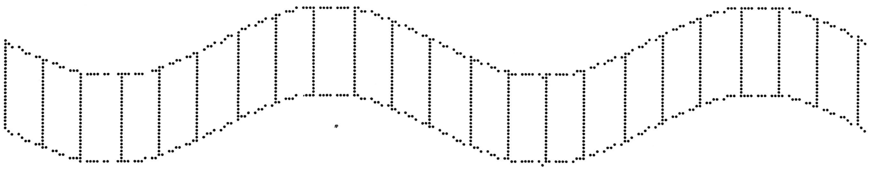


Figure 6.30

7. Draw a "twisted thread" by a pair of sine waves exactly "out of phase" (that is, with relative x shift of π; see Figure 6.31).

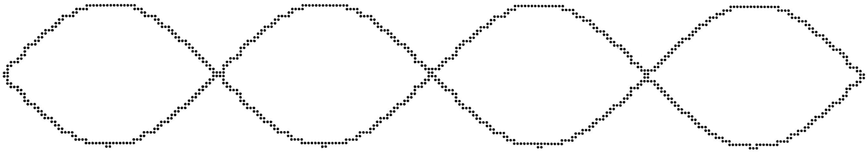


Figure 6.31

8. Shade the area between the waves (Figure 6.32).

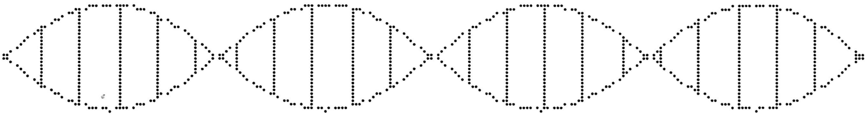


Figure 6.32

9. Draw a sine wave, and another with double or treble the frequency, and double the amplitude (Figure 6.33).

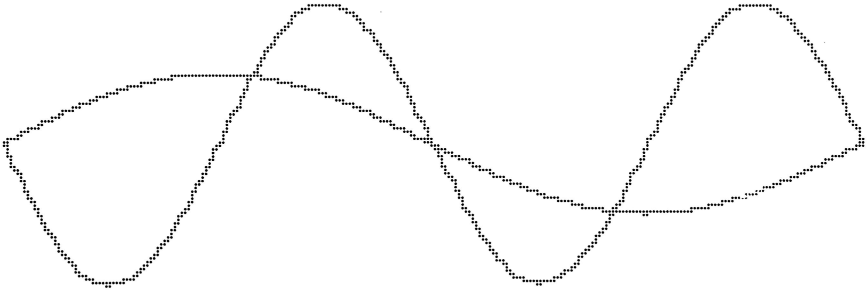


Figure 6.33

10. Experiment with variations of suggestion (9) that form interesting patterns.

11. Draw two sine waves and the curve that is obtained by adding the *y* values. Show what happens when the amplitudes are equal and the relative shift is 0, and then when it is π.
12. "Rectify" a sine wave; that is, plot the absolute value of *y*.
13. Vary some of the earlier suggestions to use rectified sine waves.
14. Plot cosines instead of sines. See how they are related in the displays that result.
15. Streamline the programs by precomputing tables of sines just once, from the SIN function.
16. Streamline the program using the formulas for sin($a+b$) and cos($a+b$), if you like trigonometry.

The SINEWAVE program. Note: omit REMs as explained on page 124.

```
100  REM *** SINEWAVE ***
110  LOMEM: 16384
120  IM=7: REM *** NUMBER OF PRESET SINE WAVES ***
130  REM *** AMPLITUDE, FREQUENCY, SHIFTS, AND LIMITS OF PRESET CURVES ***
140  DATA 1,1,0,0,-360,360,-1,1
150  DATA .5,1,0,0,-360,360,-1,1
160  DATA 1,2,0,0,-360,360,-1,1
170  DATA 1,.5,0,0,-360,360,-1,1
180  DATA 1,1,0,270,-360,360,-1,1
190  DATA .5,1,.5,0,-360,360,-1,1
200  DATA 2,3,40,30,-200,40,36,44
210  W=5:H=4:XI=80:CF=.0174532: REM *** FONT & PLOT DATA ***
220  TEXT : HOME : HGR: HCOLOR=3
230  VTAB 21: CALL -958
240  PRINT "TYPE 1 FOR SAMPLE SINE WAVES,"
250  PRINT " 2 TO SPECIFY THE PARAMETERS,"
260  INPUT " 3 TO QUIT: ";OP
270  ON OP GOTO 290,350,1130
280  PRINT "INVALID OPTION": GOTO 230
290  IQ=0: REM *** START THE CYCLE THROUGH PRESET VALUES ***
300  VTAB 21: CALL -958
310  IF IQ>=IM THEN IQ=0: RESTORE
320  IQ=IQ+1
330  READ A,F,XS,YS,XV,XH,YV,YH
340  GOTO 470
350  VTAB 21: CALL -958: REM *** ALLOW USER TO TYPE THE PARAMETERS ***
360  PRINT "AMPLITUDE=";A;", FREQUENCY=";F
370  INPUT "NEW VALUES: ";A,F
380  VTAB 21: HTAB 11: CALL -958: PRINT A;", FREQUENCY=";F
390  PRINT "X SHIFT=";XS;", Y SHIFT=";YS
400  INPUT "NEW VALUES: ";XS,YS
410  VTAB 22: HTAB 9: CALL -958: PRINT XS;", Y SHIFT=";YS
420  PRINT "X LIMITS: ";XV;", ";XH;", Y LIMITS: ";YV;", ";YH
430  INPUT "NEW VALUES: ";XV,XH,YV,YH
440  VTAB 23: HTAB 11: CALL -958
```

```
450  PRINT XV;", ";XH;", Y LIMITS: ";YV;", ";YH
460  HOME
470  REM *** DISPLAY THE EQUATION ***
480  XL=0:YL=0
490  GOSUB 5390: REM *** DISPLAY Y ***
500  GOSUB 5370:REM  *** DISPLAY=***
510  IF YS=0 THEN 550
520  REM *** DISPLAY THE Y SHIFT ***
530  ST$=STR$(YS): GOSUB 4500
540  IF A>0 THEN  GOSUB 5350: REM *** + SIGN FOR AMPLITUDE ***
550  IF A<>-1 THEN 570
560  GOSUB 5330: GOTO 590: REM *** - SIGN FOR AMPLITUDE ***
570  IF A=1 THEN  GOTO 590: REM *** A IS 1 ***
580  ST$=STR$(A): GOSUB 4500: REM *** A IS NEITHER 1 NOR -1 ***
590  ST$="SIN(": GOSUB 4500: REM *** DISPLAY SIN( ***
600  IF F<>-1 THEN 620
610  GOSUB 5330: GOTO 640: REM *** F IS -1 ***
620  IF F=1 THEN 640
630  ST$=STR$(F): GOSUB 4500: REM *** DISPLAY F ***
640  GOSUB 5420: REM *** DISPLAY X ***
650  IF XS=0 THEN 680
660  IF XS>0 THEN  GOSUB 5350: REM *** + FOR X SHIFT ***
670  ST$=STR$(XS): GOSUB 4500: REM *** DISPLAY X SHIFT ***
680  GOSUB 5550: REM *** DISPLAY ) ***
690  REM *** PLOT THE AXES ***
700  IF XV<0 AND XH>0 THEN XA=40-200*XV/(XH-XV)
710  IF XV>0 OR XH<0 THEN XA=40
720  HPLOT XA,0 TO XA,150
730  IF YV<0 AND YH>0 THEN YA=-150*YV/(YH-YV)
740  IF YV>0 OR YH<0 THEN YA=150
750  HPLOT 40,YA TO 240,YA
760  ST$=STR$(YH): REM *** PLOT THE Y TICK MARKS AND LABELS ***
770  HPLOT XA,0 TO XA+5,0
780  XL=XA-W*LEN(ST$)-2
790  YL=1: GOSUB 4500
800  ST$=STR$(YV)
810  HPLOT XA,150 TO XA+5,150
820  XL=XA-W*LEN(ST$)-2
830  YL=148-H: GOSUB 4500
840  HPLOT 40,YA TO 40,YA-5
850  ST$=STR$(XV): REM *** PLOT THE X TICK MARKS AND LABELS ***
860  XL=40:YL=YA+2: GOSUB 4500
870  HPLOT 240,YA TO 240,YA-5
880  ST$=STR$(XH)
890  XL=240-W*LEN(ST$):YL=YA+2: GOSUB 4500
900  SX=200/(XH-XV):SY=150/(YH-YV): REM *** TO GO FROM USER TO SCREEN COORDS ***
910  FOR XU=XV TO XH STEP (XH-XV)/XI: REM *** PLOT THE SINE CURVE ***
920  YU=YS+A*SIN(CF*(F*XU+XS))
930  XN=40+SX*(XU-XV):YN=150-SY*(YU-YV)
```

```
 940  IF XU=XV THEN 970
 950  IF 40>XO OR XO>240 OR 0>YO OR YO>150 OR 40>XN OR XN>240 OR 0>YN OR YN>150 THE
      N 970
 960  HPLOT XO,YO TO XN,YN
 970  XO=XN:YO=YN
 980  NEXT XU: REM *** END OF PLOTTING CYCLE ***
1000  VTAB 21: CALL -958
1010  IF OP=2 THEN 1070
1020  PRINT "TYPE 1 FOR ANOTHER SAMPLE SINE WAVE,"
1030  PRINT " 2 TO INPUT YOUR OWN PARAMETERS,"
1040  INPUT " 3 TO QUIT: ";OQ
1050  ON OQ GOTO 310,350,1130
1060  PRINT "INVALID OPTION": GOTO 1020
1070  PRINT "TYPE 1 TO INPUT A NEW SET OF PARAMETERS"
1080  PRINT " 2 FOR A PRESET SAMPLE,"
1090  INPUT " 3 TO QUIT: ";OQ
1100  ON OQ GOTO 350,290,1130
1110  PRINT "INVALID REQUEST - PRESS A KEY"
1120  GOTO 1090
1130  HGR: TEXT : HOME : END : REM *** QUIT ***
4500  REM *** DISPLAY A STRING ***
4510  FOR I=1 TO LEN(ST$)
4530  CH$=MID$(ST$,I,1)
4540  IF CH$="0" THEN  GOSUB 5010
4550  IF CH$="1" THEN  GOSUB 5030
4560  IF CH$="2" THEN  GOSUB 5060
4570  IF CH$="3" THEN  GOSUB 5100
4580  IF CH$="4" THEN  GOSUB 5140
4590  IF CH$="5" THEN  GOSUB 5170
4600  IF CH$="6" THEN  GOSUB 5200
4610  IF CH$="7" THEN  GOSUB 5220
4620  IF CH$="8" THEN  GOSUB 5240
4630  IF CH$="9" THEN  GOSUB 5280
4640  IF CH$="." THEN  GOSUB 5310
4650  IF CH$="-" THEN  GOSUB 5330
4660  IF CH$="E" THEN  GOSUB 5440
4670  IF CH$="(" THEN  GOSUB 5530
4680  IF CH$="S" THEN  GOSUB 5470
4690  IF CH$="I" THEN  GOSUB 5490
4700  IF CH$="N" THEN  GOSUB 5510
4750  NEXT I
4760  RETURN
5000  REM *** PLOT 0 THROUGH 9 ***
5010  HPLOT XL,YL TO XL+2,YL TO XL+2,YL+4 TO XL,YL+4 TO XL,YL
5020  XL=XL+W: RETURN
5030  HPLOT XL,YL+1 TO XL+1,YL TO XL+1,YL+4
5040  HPLOT XL,YL+4 TO XL+2,YL+4
5050  XL=XL+W: RETURN
5060  HPLOT XL,YL TO XL+1,YL
5070  HPLOT XL+2,YL+1 TO XL,YL+3
```

```
5080  HPLOT XL,YL+4 TO XL+2,YL+4
5090  XL=XL+W: RETURN
5100  HPLOT XL,YL TO XL+1,YL: HPLOT XL+2,YL+1
5110  HPLOT XL+1,YL+2: HPLOT XL+2,YL+3
5120  HPLOT XL,YL+4 TO XL+2,YL+4
5130  XL=XL+W: RETURN
5140  HPLOT XL,YL TO XL,YL+2 TO XL+2,YL+2
5150  HPLOT XL+2,YL TO XL+2,YL+4
5160  XL=XL+W: RETURN
5170  HPLOT XL+2,YL TO XL,YL TO XL,YL+2 TO XL+1,YL+2
5180  HPLOT XL,YL+4 TO XL+1,YL+4: HPLOT XL+2,YL+3
5190  XL=XL+W: RETURN
5200  HPLOT XL,YL TO XL,YL+4 TO XL+2,YL+4 TO XL+2,YL+2 TO XL,YL+2
5210  XL=XL+W: RETURN
5220  HPLOT XL,YL TO XL+2,YL TO XL,YL+4
5230  XL=XL+W: RETURN
5240  HPLOT XL+1,YL TO XL+2,YL TO XL+2,YL+1
5250  HPLOT XL,YL+1: HPLOT XL+1,YL+2: HPLOT XL,YL+4
5260  HPLOT XL+1,YL+4 TO XL+2,YL+4 TO XL+2,YL+3
5270  XL=XL+W: RETURN
5280  HPLOT XL,YL+4 TO XL+2,YL+4 TO XL+2,YL TO XL,YL TO XL,YL+2 TO XL+2,YL+2
5290  XL=XL+W: RETURN
5300  REM *** PLOT  -+=YXE ***
5310  HPLOT XL+2,YL+4
5320  XL=XL+W: RETURN
5330  HPLOT XL,YL+2 TO XL+2,YL+2
5340  XL=XL+W: RETURN
5350  HPLOT XL,YL+2 TO XL+2,YL+2: HPLOT XL+1,YL+1 TO XL+1,YL+3
5360  XL=XL+W: RETURN
5370  HPLOT XL+1,YL+1 TO XL+3,YL+1: HPLOT XL+1,YL+3 TO XL+3,YL+3
5380  XL=XL+W: RETURN
5390  HPLOT XL,YL TO XL+2,YL+2 TO XL+4,YL
5400  HPLOT XL+2,YL+4 TO XL+2,YL+2
5410  XL=XL+W: RETURN
5420  HPLOT XL,YL+4 TO XL+4,YL: HPLOT XL,YL TO XL+4,YL+4
5430  XL=XL+W+2: RETURN
5440  HPLOT XL+2,YL TO XL,YL TO XL,YL+4 TO XL+2,YL+4: HPLOT XL,YL+2 TO XL+2,YL+2
5450  XL=XL+W+1: RETURN
5460  REM *** PLOT SIN() ***
5470  HPLOT XL+2,YL TO XL,YL TO XL,YL+2 TO XL+2,YL+2 TO XL+2,YL+4 TO XL,YL+4
5480  XL=XL+W: RETURN
5490  HPLOT XL,YL TO XL+2,YL:HPLOT XL,YL+4 TO XL+2,YL+4:HPLOT XL+1,YL TO XL+1,YL+4
5500  XL=XL+W: RETURN
5510  HPLOT XL+2,YL TO XL+2,YL+4 TO XL,YL TO XL,YL+4
5520  XL=XL+W: RETURN
5530  HPLOT XL+1,YL+4 TO XL,YL+2 TO XL+1,YL
5540  XL=XL+W: RETURN
5550  HPLOT XL,YL+4 TO XL+1,YL+2 TO XL,YL
5560  XL=XL+W: RETURN
```

Chapter 7

Some further directions

Every topic in computer graphics opens up the discussion of many more. This chapter touches on three that will be continued in further volumes along with other topics—perspective drawings of three-dimensional objects, the use of color in animation as well as in static pictures, and the mathematics used to make computer graphics easier to program and faster to execute.

7.1 ADDING A LITTLE PERSPECTIVE

We now take a step into the third dimension, using the program ZVIEW to produce pictures like Figure 7.1. The program prompts for the description of a cube or any other solid with straight edges and flat (plane) faces (that is, a polyhedron). ZVIEW behaves as if the center of the screen is the origin of an xyz coordinate system, with a horizontal x axis, a vertical y axis, and the z axis pointing through the center of the screen away from the viewer (Figure 7.2).

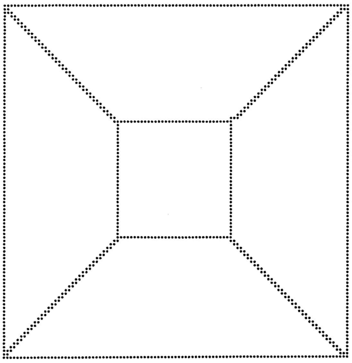


Figure 7.1

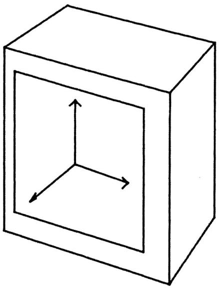


Figure 7.2

The program assumes that you are looking at the screen along a line through its center, from a distance that is five times its height, as shown in Figure 7.3.

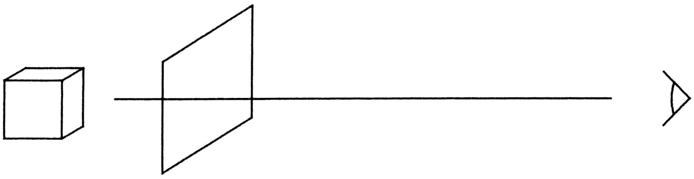


Figure 7.3

By simple proportionality, the display can also be regarded as the scaled picture of a projection onto a screen of a different height with the eye five times this new height away from it.

You define a solid for ZVIEW by typing

1. The coordinates of the corners in any order. The program numbers these 1, 2, 3, ... in the order they are typed, unless you wish to give them other identifying numbers.
2. For each edge, the numbers that identify the corners at each end.

The program begins by displaying the menu:

```
TYPE:
1 TO DEFINE DISTANCES, 2 TO DEFINE POINT,
3 TO DISPLAY EDGE, 4 TO CLEAR SCREEN,
5 TO RESTART, 6 TO QUIT:
PLEASE TYPE OPTION:
```

If you type 1, the program displays the secondary menu:

```
TYPE 1 FOR POSITION OF EYE,
     2 FOR DISTANCE TO SCREEN,
     3 FOR HEIGHT OF SCREEN,
     IN GLOBAL DIMENSIONS:
```

After you type the appropriate number, the program displays the current value of the dimension you requested. You type the new value, and the main menu is displayed again.

To specify the coordinates of a corner, you type 2 when the main menu is on the screen. The program prompts for the identification number of the point, and displays the lowest number that has not been used so far, in parentheses. You respond, and the program prompts for the coordinates. To define an edge, you type 3 when the main menu is on the screen. The program prompts for the identification numbers of its ends. After you have typed these, it displays the coordinates of the points and allows you to draw the edge or to skip it. Option 4 lets you clear the screen without erasing the coordinates that have been stored. Options 5 and 6 let you restart the entire process or quit, respectively.

The underlying geometry is very simple (Figure 7.4). The diagram shows a pin of length L, projecting from the z axis and parallel to the x axis. Its perspective projection on the plane of the screen is the image that obscures it, when observed from E. If this has length l, the simple proportionality rule for similar triangles gives:

$$\frac{l}{D}=\frac{L}{(R+D)} \qquad \text{hence} \qquad l=\frac{L}{(1+R/D)}$$

where R and D are the distances of the pin and the eye from the origin O. Now consider a pin of length H, projecting vertically from the z axis, and parallel to the y axis (Figure 7.5). The height, h, of its perspective projection is given correspondingly by

$$h=\frac{H}{(1+R/D)}$$

Use the symbols (x_s, y_s) for the coordinates of a point in a perspective projection on the Oxy plane. Then l is the x_s value of the projection of a point with x and z coordinates (in the Oxyz system) equal to L and R. Also h is the y_s value of the projection of a point with y and z coordinates (in the Oxyz system) equal to H and R.

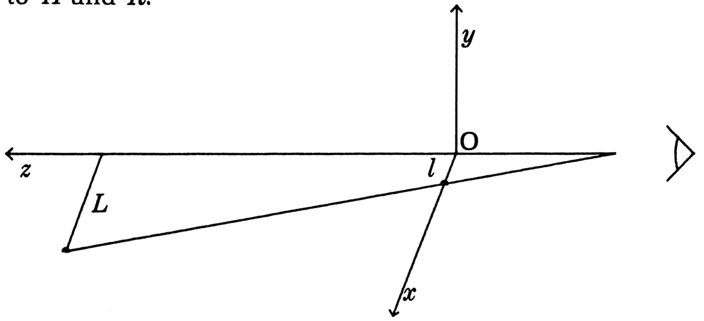


Figure 7.4

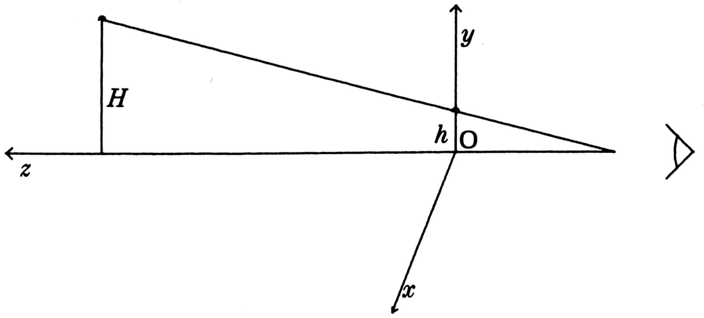


Figure 7.5

It follows that a point with coordinates (x, y, z) projects into the point on the screen, with coordinates

$$x_s = \frac{x}{(1+z/D)}, \qquad y_s = \frac{y}{(1+z/D)}$$

These are the basic equations for perspective transformations along the z axis that are used in a direct manner by the program. This cycles through a FOR loop to transform the coordinates of each corner of the object in turn. Another FOR loop then cycles through the matrix of edge definitions, and draws the appropriate lines.

The main variables in the program are

`XG(50)`	The x coordinate of an apex
`YG(50)`	The y coordinate of an apex
`ZG(50)`	The z coordinate of an apex
`IM%`	The maximum number of apexes allowed by the program
`ID%`	The identification number of an apex
`ZE`	The distance from the eye to the origin
`D`	The distance from the eye to the screen
`S`	The height of the screen
`F`	The factor to convert the x, y coordinates of an apex to the screen coordinates of its projection
`I1%`	The identification number of the point at one end of the edge that is being drawn
`I2%`	The identification number of the point at the other end

Suggested variations

1. Let the user describe a collection of rectangular and cubic bricks, resting on the Oxz plane, by giving the height and the coordinates of the three points on the base of each, instead of the complete set of corners and edges.
2. Deal correspondingly with a collection of pyramids.

The ZVIEW program

```
100  REM *** ZVIEW FOR PERSPECTIVE PROJECTION ALONG Z AXIS ***
110  DIM XG(50),YG(50),ZG(50)
120  IM%=50:ID%=1
130  ZE=20:D=5:S=15
140  HOME : HGR : HCOLOR=3: POKE 34,20
150  HOME : PRINT "TYPE: "
160  PRINT "1 TO DEFINE DISTANCES, 2 TO DEFINE POINT";
170  PRINT "3 TO DISPLAY EDGE, 4 TO CLEAR SCREEN,"
180  INPUT "5 TO RESTART, 6 TO QUIT: ";OP
190  HOME : ON OP GOTO 220,350,440,140,120,610
200  PRINT : FLASH : PRINT "INVALID OPTION: ";OP;", PRESS ANY KEY"
210  NORMAL : GET A$: GOTO 150
220  HOME : PRINT "TYPE 1 FOR POSITION OF EYE,"
230  PRINT "     2 FOR DISTANCE TO SCREEN,"
240  PRINT "     3 FOR HEIGHT OF SCREEN,"
250  INPUT "     IN GLOBAL DIMENSIONS: ";OQ
```

```
260  HOME : ON OQ GOTO 290,310,330
270  PRINT : FLASH : PRINT "INVALID REQUEST: ";OQ;", PRESS ANY KEY"
280  NORMAL : GET A$: GOTO 220
290  PRINT "DISTANCE FROM EYE TO ORIGIN (NOW=";ZE;"): ";
300  INPUT "";ZE: GOTO 150
310  PRINT "DISTANCE FROM EYE TO SCREEN (NOW=";D;"): ";
320  INPUT "";D: GOTO 150
330  PRINT "HEIGHT OF SCREEN (NOW=";S;"): ";
340  INPUT "";S: GOTO 150
350  PRINT "ID # OF POINT(NEXT FREE #=";ID%;"): ";
360  INPUT "";JD%
370  IF JD%>0 AND JD%<IM% THEN 410
380  PRINT : FLASH : HTAB 10: PRINT "INVALID ID#": NORMAL
390  PRINT "PRESS ANY KEY TO RETURN TO MAIN MENU"
400  GET A$: GOTO 150
410  INPUT "COORDINATES: ";XG(JD%),YG(JD%),ZG(JD%)
420  IF JD%=ID% THEN ID%=ID%+1
430  GOTO 150
440  INPUT "ID #S OF END POINTS: ";I1%,I2%
450  IF 0>I1% OR I1%>IM% OR 0>I2% OR I2%>IM% THEN 380
460  HOME
470  PRINT "COORDINATES: ";XG(I1%); SPC( 3);YG(I1%); SPC( 3);ZG(I1%)
480  PRINT  SPC( 13);XG(I2%); SPC( 3);YG(I2%); SPC( 3);ZG(I2%)
490  INPUT "TYPE 1 TO DISPLAY, 2 TO SKIP: ";OQ
500  IF OQ<>1 THEN 150
510  F=(150*D)/(S*(ZE-ZG(I1%)))
520  X1=140+F*XG(I1%):Y1=75-F*YG(I1%)
530  IF 0<=X1 AND X1<=279 AND 0<=Y1 AND Y1<=159 THEN 550
540  PRINT "FIRST POINT OUT OF RANGE: ";X1,Y1: GOTO 390
550  F=(150*D)/(S*(ZE-ZG(I2%)))
560  X2=140+F*XG(I2%):Y2=75-F*YG(I2%)
570  IF 0<=X2 AND X2<=279 AND 0<=Y2 AND Y2<=159 THEN 590
580  PRINT "SECOND POINT OUT OF RANGE: ";X2,Y2: GOTO 390
590  HPLOT X1,Y1 TO X2,Y2
600  GOTO 150
610  TEXT : HOME : END
```

7.2 A ROTATING DONUT

The program TRYTOR goes one step further. It draws pictures of a donut viewed from different angles (Figure 7.6). The program is very short—all the formulas needed are encapsulated in a few simple statements. The donut is represented by a set of vertical disks of radius a, with their centers on a horizontal circle of radius b. This has its center at the origin of a coordinate system with a vertical y axis. The program draws a picture of the donut viewed along a line through the origin that is defined by two angles that are commonly used in solid geometry and star gazing–the polar angle and the azimuth. These are denoted as usual by the Greek letters θ and ϕ (theta and phi). The distance from the viewer to the donut is assumed to be much greater than the dimensions of the donut, so that the projection is "isometric."

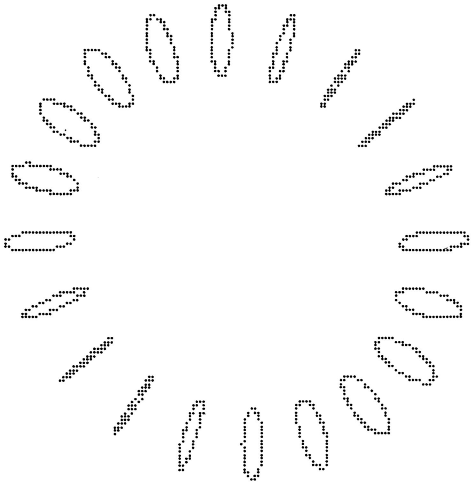


Figure 7.6A. $a=15$, $b=40$, $\theta=30$, $\phi=30$.

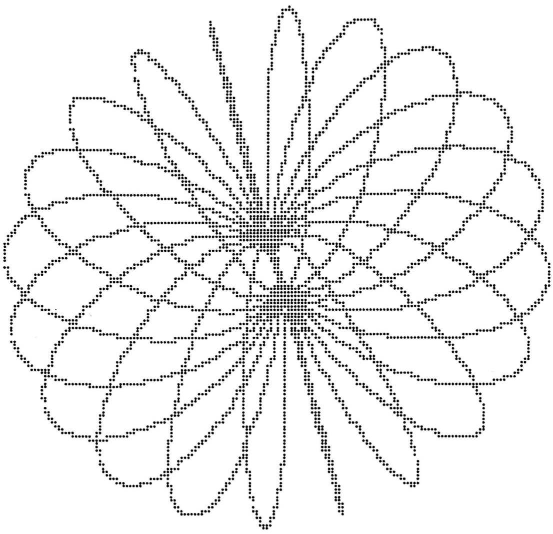


Figure 7.6B. $a=40$, $b=40$, $\theta=30$, $\phi=45$.

The program (1) prompts for the values of a, b, θ, and ϕ, (2) draws the projection of the donut, and (3) prompts the user either to request another picture or to quit.

The main variables in the program are

`X(I)`	The x coordinate of a point on the rim of one of the disks that makes up the donut, measured along an x axis through the center of the disk
`Y(I)`	The y coordinate of the point, measured parallel to the y axis of the main coordinate system
`A`	The radius of the disk
`B`	The radius of the donut
`TH,PH`	The angles θ and ϕ
`J`	Takes values 1–100 around the donut
`Z1,...,Z4,CP,SP,U1,...,U4`	Intermediate results that are used several times

The program uses values of a and b to construct the coordinates X(I) and Y(I) of the points around one of the disks, in the "local" system of the disk. The corresponding coordinates in the system that defines the view line are:

$$x=\mathrm{X(I)}\cos(\mathrm{J}/2\pi), \qquad y=\mathrm{Y(I)}, \qquad z=\mathrm{X(I)}\sin(\mathrm{J}/2\pi)$$

The coordinates (x_r, y_r) of the same point, in a system that has its z axis along the view line, are:

$$x_r=x\cos\theta\cos\phi+y\cos\theta\sin\phi-z\sin\theta$$

$$y_r=-x\sin\phi+y\cos\phi$$

$$z_r=x\sin\theta\cos\phi+y\sin\theta\sin\phi+z\cos\theta$$

Because we just want the projection onto a plane perpendicular to the view line, the (x_r, y_r) coordinates are all that is needed. The program plots these for 100 points around each of the 200 disks that make up the donut.

The program can be extended easily to provide perspective projections when the donut is viewed close up. TRYTOR draws the pictures as if the viewer's head was tilted toward one side. The pictures can be rotated on the screen to compensate for this. Several sections would be needed, however, just to start on a discussion of three-dimensional effects, which also include the shading of surfaces, the casting of shadows, the plotting of contours, and the removal of hidden lines. Consideration of these topics therefore must be deferred.

Suggested variations

1. Let the user specify the number of disks in a donut and the number of points around each.
2. Allow several donuts on the screen together.
3. Make the program range systematically through a succession of θ and ϕ values.
4. Apply the principle to other objects that can be constructed from disks.
5. Speed up the calculation of sines and cosines at the beginning of the program.

The TRYTOR program

```
100   REM *** TRYTOR PROGRAM TO DRAW TOROID ***
110   DIM CS(101),SN(101),X(101),Y(101)
120   REM *** SET UP TABLE OF SINES&COSINES ***
130   DN= 0628:AN=0
140   FOR I=1 TO 100
150   CS(I)=COS(AN):SN(I)=SIN(AN):AN=AN+DN
160   NEXT I
170   HGR : HCOLOR=3: HOME : VTAB 21
180   INPUT "ENTER A,B,THETA,PHI: ";A,B,TH,PH
190   REM *** CREATE PROTOTYPE CIRCLE ***
200   FOR I=1 TO 100
210   X(I)=B+A*CS(I):Y(I)=A*SN(I)
220   NEXT I
230   REM *** FORMSIN&COSOF THETA&PHI ***
240   Z1=COS(TH):Z2=SIN(TH):Z3=COS(PH):Z4=SIN(PH)
250   REM *** FORM SUCCESSIVE RINGS ***
260   FOR J=1 TO 100 STEP 5
270   CP=CS(J):SP=SN(J)
280   U1=Z1*Z3*CP-Z2*SP:U2=Z1*Z4
290   U3=-Z4*CP:U4=Z3
295   HPLOT 140+U1*X(1)+U2*Y(1),U3*X(1)+U4*Y(1)+80
300   FOR K=1 TO 100
310   HPLOT  TO U1*X(K)+U2*Y(K)+140,U3*X(K)+U4*Y(K)+80
320   NEXT K
330   NEXT J
340   PRINT "TYPE E TO END";: GET A$: IF A$="E" THEN  END
350   PRINT
360   INPUT "CLEAR SCREEN?(Y OR N) ";A$: IF LEFT$(A$,1)="N" THEN 180
370   GOTO 170
```

7.3 A KALEIDOSCOPE OF FUN

Color graphics is another burgeoning area of interest and importance. All the programs in this book can benefit from the addition of color, but they can be understood without it. The KALEID program produces a kaleidoscope effect on a color screen. Even on a monochrome monitor it produces an interesting pattern of constantly changing form . Figure 7.7 depicts the different colors by different symbols. Only low-resolution graphics are used. Each of the 40×40 blocks is made to change color (or shading in monochrome) rapidly, in a cyclic and symmetrical manner. Two nested FOR loops cycle through the blocks and set the COLOR number to a sine or cosine function of the block coordinates, truncated to an integer. The SCRN function is used to find the coordinates of a block. The loops are indexed by C and D, the only variables in the program.

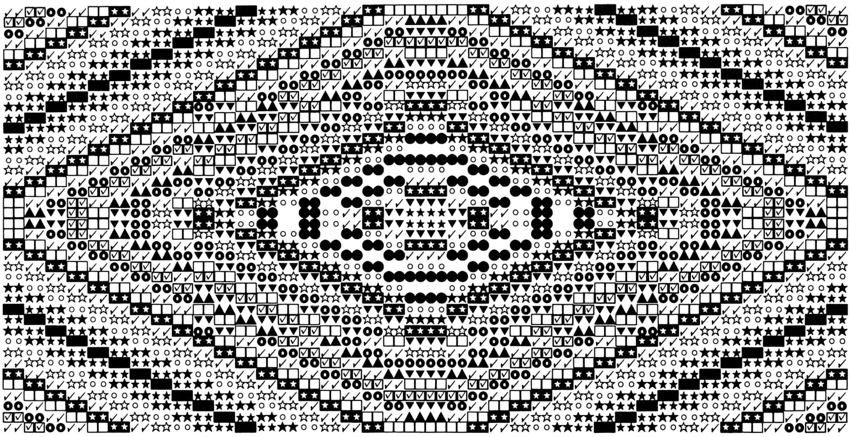


Figure 7.7

Suggested variations

1. Apply the principle to areas of different shapes.
2. Produce corresponding effects with different symmetries.

The KALEID program

```
100  REM *** KALEID-USES LO-RES GRPAHICS TO CREATE COLORFUL PATTERNS ***
110  GR
120  REM *** PROGRESS INWARD TO CENTER  OF SCREEN ***
130  FOR C=0 TO 19
140  FOR D=C TO 20
150  COLOR=ABS(15*SIN(C+D)): GOSUB 340
160  NEXT D,C
170  REM *** PROGRESS TO EDGES OF SCREEN ***
180  FOR C=19 TO 0 STEP -1
190  FOR D=C TO 20
200  COLOR=ABS(15*COS(C+D)): GOSUB 340
210  NEXT D,C
```

```
220  REM *** GO BACK IN ***
230  FOR C=0 TO 19
240  FOR D=C TO 20
250  COLOR=15- SCRN( C,D): GOSUB 340
260  NEXT D,C
270  REM *** COME BACK OUT ***
280  FOR C=19 TO 0 STEP -1
290  FOR D=C TO 20
300  COLOR= SCRN( C,D)+1: GOSUB 340
310  NEXT D,C
320  GOTO 230: REM *** CONTINUE CYCLE ***
330  END
340  REM *** DRAW 8 POINTS, WITH SYMMETRY AROUND THE SCREEN'S CENTER ***
350  PLOT C,D: PLOT D,C: PLOT 39-C,D: PLOT 39-D,C
360  PLOT 39-C,39-D: PLOT 39-D,39-C: PLOT C,39-D: PLOT D,39-C
370  RETURN
```

7.4 MORE SIMPLE GEOMETRY

Personal graphics provides considerable scope for the elementary geometry that is taught at school today, and also for some that was taught 50 years ago, but has had little mention recently. For example, you probably learned the slope-intercept equation for a straight line:

$$y = mx + b$$

where m is its slope and b is the intercept on the y axis (Figure 7.8). You may even have used this to write programs. Unfortunately, it cannot be used when a line is vertical, so you have to treat this as a special case. You can write another equation for a line, however, that contains three quantities that define it completely, and which can always be used. This "normal equation" is simply

$$Ax + By + C = 0$$

where A and B are the cosine and sine of the angle between the x axis and a perpendicular from the origin to the line and C is the length of the perpendicular (see Figure 7.9). But of this and many other matters we can only say "More anon."

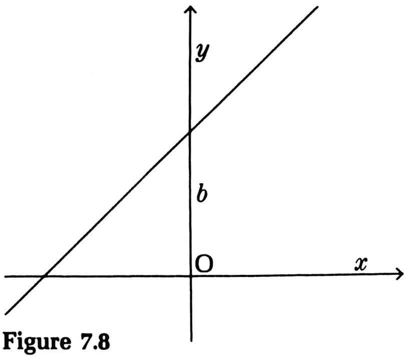


Figure 7.8

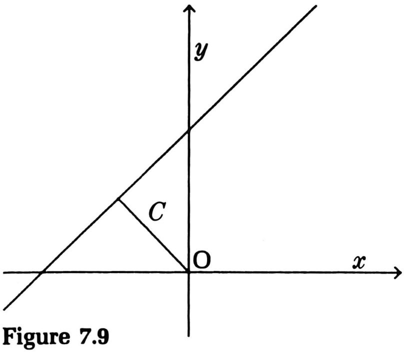


Figure 7.9

Chapter 8

A short account of Applesoft

8.1 LANGUAGES FOR THE APPLE

Programs for the Apple can be written in several languages. Applesoft—which we use in this book—and integer BASIC are derived from the original Dartmouth College BASIC. The Apple can also be programmed in 6502 assembly language and, with additional circuit cards, in PASCAL and FORTRAN. There are powerful operating systems—DOS and CP/M—and many proprietary programs for particular applications. Backup material for the programs in this book can be found in the following references:

[1] *The Applesoft Tutorial* [Apple Product Number A2L0018(030-0044-C)]
[2] *Apple II Reference Manual* [Apple Product Number A2L0001A(030-0004-C)]
[3] *BASIC Programming Reference Manual* [Apple Product Number A2L0006(030-0013-E)]
[4] *The DOS Manual* [Apple Product Number A2L0036(030-0115-B)]

8.2 APPLESOFT PROGRAMS

Applesoft is a powerful extension of the original BASIC. A program is typed on one or more "lines," which can be up to 255 characters long. Each starts with a line number in the range 0–65,536. Several statements separated by colons can be typed on a line. A line is ended by typing a carriage return. It may "fold" on the screen, but this is irrelevant. A "remark" statement begins REM and continues to the end of the line as just defined.

Constants and variables may be real numbers, integers, or strings. An identifier may be either a letter (for example, X) or a letter followed by another letter or a digit (for example, AB, G7), and then also a % sign for an integer (for example, Q%, PG%, H5%) or a $ sign for a string (for example, D$, EZ$, F2$). String constants are displayed between double quotation marks (for example, "EUREKA"). Strings are varying length, up to 256 characters long. Arrays of one or more dimensions are declared in DIM statements, for example,

```
DIM A(5),E%(10,7),G%(3,5),B(2,4,3)
```

Subscript values start at 0; that is, this statement allows A(0)–A(5) and E%(0,0) –E%(10,7) to be used. The maximum number of dimensions is 88. The same identifier can be used for an array and for a nonsubscripted variable. (For example, A, E%, G$, and B can be used in the program that contains the DIM statement just shown.) Arrays of integer variables occupy less space than arrays of real variables. Identifiers are allowed that are longer than just described, but the system elides redundant characters. (For example, SCALE, LIMIT%, and TITLE$ are treated as SC, LI%, TI$.) The use of identifiers instead of constants within a loop increases the speed of execution.

8.3 INITIALIZATION AND ASSIGNMENT

All the numerical and string variables in a program are initialized to 0 and to null strings, respectively, when execution begins. The statement CLEAR repeats this initialization. Variables are given values by

1. Assignment statements that use the = symbol and may begin with the word LET (for example, X=5: LET Y=7)
2. Input statements (discussed in Section 8.6)
3. DATA statements of the form

`DATA` list-of-values-separated-by-commas

These are assigned by statements of the form

`READ` list-of-names-of-variables

For example, if a program begins with

```
100 DATA 1,2,3,4,5
110 READ A,B,C,D,E
```

it sets the variables A–E to the values 1–5. The program moves sequentially through (1) the variables in the READ statements that it executes and (2) the values in the DATA statements in the order that these occur in the program. This is completely free and independent of the flow of control. Thus

```
100 READ A,B: DATA 1,2,3: READ C
110 DATA 4
120 DATA 5: READ D,E
```

has the same result as the previous example. This is unaffected by further statements between lines 100, 110, and 120, provided these do not contain READ or DATA statements.

READing is restored to the start of the first DATA statement by the statement

```
RESTORE
```

8.4 NUMERIC AND STRING EXPRESSIONS

The symbols +, −, *, / and ^ (shift N) are used for addition, subtraction, multiplication, division, and exponentiation in arithmetic expressions.

If *x* denotes a numerical expression, the function INT(*x*) returns the "floor" of *x* (for example, when *x* is 1.5, 3, −2.5, −4, then INT(*x*) is 1, 3, −3, −4, respectively.) The function SGN(*x*) returns −1, 0, and 1, respectively, when *x* is negative, zero, and positive. ABS(*x*) and SQR(*x*) return the absolute value and the square root of *x*. The function RND(*x*) returns a random number in the range 0 to 0.999...9. This is different from the previous number returned by RND if *x* is positive. It is the same if *x* is zero. It is determined uniquely by *x* if *x* is negative. The functions SIN, COS, and TAN return, respectively, the sine, cosine, and tangent of an angle expressed in radians (that is, the angle in degrees times $\pi/180=0.0174532$). The function ATN returns the arctangent, in the range $-\pi/2$–$\pi/2$. The result is in radians. To convert to degrees, multiply by 57.2958. The functions EXP(*x*) and LOG(*x*) return the exponential and natural logarithm of *x*, respectively.

The function STR$(*x*) converts a real or integer argument to a string. If *a*$ denotes a string expression, VAL(*a*$) returns the real or integer value that *a*$ represents, starting at its leftmost character and proceeding as far to the right as valid. [For example, VAL("123GO") returns the integer 123.] The function ASC(*a*$) returns the ASCII code (as an integer) of the first character in *a*$. The function CHR$(*n*), where *n* is an integer argument in the range 0–255, returns the character that has *n* as its ASCII code. If *a*$, *i*, *j* denote a string expression and two numeric expressions, then LEN(*a*$), LEFT$(*a*$,*i*), RIGHT$(*a*$,*i*), MID$(*a*$,*i*), and MID$(*a*$,*i*,*j*) return the length, the left and the right *i* characters of *a*$, and the substrings that start at character *i* and extend to the end of the string and for just *j* characters, respectively.

A function can be defined by a statement of the form

```
DEF FN function-name(dummy-argument)=expression
```

The expression on the right hand side contains the dummy argument. Only the first two characters of the function name are recognized by the system. As an example, a function that returns the modulus of a variable X with respect to 10 (that is, the remainder after dividing by 10 when X is positive, and the negative of the modulus of −X, when X is negative) can be defined by

```
DEF FN MD(X)=INT((X/10-INT(X/10))*10+.05)*SGN(X/10)
```

The statements

```
A=FN MD(17): B=FN MD(-25)
```

then set A to 7 and B to −5. The function MD can be expanded to give the modulus of X with respect to a second variable Y by replacing 10 by Y throughout.

The function PEEK(*m*) returns the contents of memory location *m*, as an integer. This is used extensively, to test the current values of words that the system uses for special purposes. The command POKE *m,n* sets memory location *m* to the value of *n*; *m* and *n* denote integer expressions.

8.5 FLOW OF CONTROL

The following statements affect the flow of control:

1. `GO TO` line-number
   (GO TO also can be typed GOTO.)
2. `ON` integer-expression `GOTO` line-number-list
   (The program "falls through" if the integer expression is not in the range 1 to the length of the line-number-list.)
3. `GOSUB` line-number
   (This branches to a subroutine that is part of the overall program; you must assign values of the parameters before branching, and assign the results after returning.)
4. `ON` integer-expression `GOSUB` line-number-list
5. `RETURN`
6. `POP`
   (This makes the next RETURN statement jump back an extra level in the stack of return addresses.)
7. `FOR` index=starting-index `TO` ending-index [`STEP` increment]
   [This begins an indexed cycle (DO loop in many languages); expressions can be used for the limits and/or increment.]
8. `NEXT`
   (This ends an indexed cycle; one or more indexes can be typed after the word NEXT—this increases clarity and reduces speed.)
9. `CALL` absolute-address-of-system-subroutine
   (A few are mentioned in this section.)
10. `ONERR GO TO` line-number
    [The line number in which the error occurred then is given by

    ```
    PEEK(218)+PEEK(219)*256
    ```

    A numerical code that identifies the type of error is given by

    ```
    PEEK(222)
    ```

    These include 5 (end of input file), 42 (out of data for a READ), 53 (illegal quantity), 69 (overflow), 77 (bad subscript), 133 (division by zero), 176 (string too long) and 254 (bad input). A longer list is given in reference [3], p. 136.]
11. `RESUME`
    (After a branch on an ONERR statement, this sends control to the instruction that follows the one that caused the error.)

12. `IF` condition `THEN` executable-statement
If the executable statement is a GO TO, the words GO TO can be omitted. If the condition is a numerical expression, it is considered true if nonzero, and false if precisely zero. Arithmetic and string comparisons refer to relative numerical value and filing sequence, respectively, and use the symbols <, < =, =, > =, and > with the obvious meanings, and < > for "not equal to". Conditions can be compounded using the logical operators AND, OR, and NOT and parentheses. When a condition is satisfied, any further statements in the program line are executed too. Thus if X is 5,

```
IF X=5 THEN A=2: B=3: PRINT C
```

sets A to 2, B to 3, and prints C. This feature is very useful, but a possible trap when debugging, for people who think in terms of some other languages.

8.6 INPUT/OUTPUT INSTRUCTIONS

A statementof the form

`PRINT` data-list

sends output to the monitor, unless (1) another device has been requested by the PR# statement described later in this section or (2) output is being written to disk as described in Section 8.8. The statement starts a new line (except when the previously executed PRINT statement ends with a semicolon). The data list may be null (just to leave a blank line, or to cause a short delay) or it may contain one or more numerical and/or string expressions. A semicolon (;) separates items that are to be concatenated without intervening space in the output. Ending a PRINT statement with a semicolon makes the next PRINT statement start printing in the next character position. A comma (,) in a PRINT statement separates items that are "tabbed" to positions 1, 16, and 32 in succession.

The formatting functions SPC(*i*) and TAB(*i*), also can be used in a data list. These leave *i* spaces and tab to position *i*, respectively (where *i* is a valid numeric expression). The commands

`HTAB` *i*   `VTAB` *i*   `HOME`

move the cursor to, respectively, horizontal position *i* on the current line (or if it has been passed, on the next line), the start of the line *i*, and the start of the first line. HOME also clears the screen. In the HTAB and VTAB commands, *i* is 1 for the first line and the first character, respectively.

The function

`POS`(dummy-expression)

returns the horizontal position of the cursor. The dummy expression is needed to separate the parentheses, but it must be legal (for example, 0). The horizontal position of the cursor also is given by PEEK(36), and the vertical position by PEEK(37). These count the characters and lines starting at 0, in contrast to the HTAB and VTAB commands that start counting at 1.

The commands INVERSE, FLASH, and NORMAL switch the appearance of further characters into, respectively, the "inverse video" (black-on-white), "flashing" (alternating white-on-black and black-on-white), and "normal" (white-on-black) modes.

The command

```
SPEED=expression-with-integer-value-between-0-and-255
```

sets the speed with which characters are output.

A statement of the form

```
PR#n
```

where *n* stands for an expression with an integer value 1–7 routes output to the device, such as a printer, that is connected to the slot at the back of the Apple that this integer specifies. The statement PR#0 restores output to the screen.

A statement of the form

```
INPUT ["prompt"];data-list
```

starts a new line on the monitor, which displays the prompt (or a question mark if the prompt and the quote marks were omitted). It then assigns, to the successive identifiers in the data list, successive numbers and strings typed by the user, unless (1) another input device has been requested by an IN# statement analogous in form and action to the PR# statement or (2) input is being read from disk (see Section 8.9). Commas separate the items in the data list and the values in the input stream.

An array can be saved on a tape cassette attached to the Apple by the statement

```
STORE array-name
```

It is retrieved by the statement

```
RECALL array-name
```

The INPUT statement provides a basic method of making interactive programs wait until the user types some input. A program also can be made to wait, without displaying a prompt, by a statement of the form

```
GET a$
```

where *a$* denotes a string variable. When any key is pressed, *a$* is set to the character that it represents and the program continues. If a key was pressed before the GET statement was executed, *a$* is set to the character that it represents, and the program continues without any delay.

The WAIT command (reference [3], p. 41) provides more elaborate ways of controlling a delay. You can also let a program continue until a key is pressed —the opposite effect to the GET command. The statement

```
POKE -16368,0
```

sets to zero the value returned by

```
PEEK (-16384)
```

When a key is pressed, the value of this PEEK changes to the ASCII code that corresponds to the key. The codes for all the printable characters are listed in Section 5.2. The full list, which includes control codes, is given in reference [1], p. 138. The values in that table must be increased by 96 for use in Applesoft.

You can set (1) the left margin of the window in which text is displayed and scrolled, (2) the width of the window, (3) its top margin, and (4) its bottom margin by commands of the form

```
POKE 32,L   POKE 33,W   POKE 34,T   POKE 35,B
```

where L, W, T, and B stand for the required values. The command HOME operates just within the window that these POKEs control or, by default, within the full screen.

After repositioning the cursor (using VTAB and HTAB), you can erase material to the end of the current line or to the end of the text by

`CALL -958` and `CALL -868`

respectively. The same effects are produced by

`PRINT CHR$(27);"F"` and `PRINT CHR$(27);"E"`

These send the escape F and escape E codes to the output mechanism.

Paddle input is registered by the commands PDL(1) and PDL(2). Audio output is provided basically by repeating the statement

```
X=PEEK(-16336)
```

which produces a click on the speaker. Tone is determined by the interval between clicks.

8.7 GRAPHICS COMMANDS

The command

```
GR
```

switches the Apple into the low-resolution graphics mode. This provides a 40×40 raster of blocks on the screen that can be addressed by the commands of the following forms, in which x and y take values 0–39 from left to right and from top to bottom, respectively. The commands

```
PLOT x,y
HLIN x1,x2 AT y
VLIN y1,y2 AT x
```

plot a point at position x, y, draw a horizontal line from x_1 to x_2 at depth y, and draw a vertical line from y_1 to y_2 at horizontal position x. A command of the form

```
COLOR=n
```

where n is an integer expression in the range 0–15, sets the color. The system initializes the color number to 0, for black (that is, the background). The colors assigned by the different numbers in the United States are

0 black
1 magenta
2 dark blue
3 purple
4 dark green
5 grey
6 medium blue
7 light blue
8 brown
9 orange
10 grey
11 pink
12 green
13 yellow
14 aqua
15 white

On a monochrome (black-and-white) screen, different color numbers produce different shading patterns in the blocks that are lighted up.

The system normally leaves a window for four lines of text at the bottom of the screen in both the low- and high-resolution graphics modes. The low-resolution graphics area is extended downward to cover this, allowing y to take values up to 47, by the POKE command

```
POKE -16302,0
```

The effect of this is reversed by

```
POKE -16301,0
```

The command

```
SCRN(x,y)
```

returns the color code of the block with the specified coordinates. The arguments can be any numerical expressions.

The command

```
HGR
```

switches the Apple into the high-resolution graphics mode. This provides a 280×160 raster of pixels on the screen that can be addressed by commands of the following forms, in which x goes from 0 to 279 and y from 0 to 159, measured from the left and top of the screen, respectively. The commands

```
HPLOT x1,y1
HPLOT x1,y1 TO x2,y2
HPLOT x1,y1 TO x2,y2 TO ... TO xn,yn
```

plot a single point, a single vector, and an n-component polyvector, respectively.

The color is set by a command of the form

```
HCOLOR=n
```

where n is an integer expression with a value in the range 0–7. The color for the high-resolution mode is indeterminate until an HCOLOR command has been executed. The colors assigned by the different numbers are

0 black	2 blue	4 black	6 varies
1 green	3 white	5 varies	7 white

The colors 5 and 6 depend on the monitor, and the color 3 produces nonwhite dots in certain circumstances (see reference [3], p. 89). On a monochrome screen, colors 3 and 7 cause all the pixels that are addressed to light up; colors 0 and 4 cause the pixels that are addressed to be unlighted. Other colors have varying effects.

The system normally leaves a window for four lines of text at the bottom of the screen. The high-resolution graphics area is extended downward to cover this, allowing y to take values 0–191, by the command

```
POKE -16302,0
```

The effect of this is reversed by

```
POKE -16301,0
```

The command

```
HGR2
```

preserves, in the first page of high-resolution graphics, the display that has been put on the screen, using the commands just described, and allows a second 280×192 page of high-resolution graphics to be displayed by use of the HPLOT command. This is preserved as the second page of high-resolution graphics, and the first page is redisplayed by the command

```
POKE -16300,0
```

The second page is redisplayed by

```
POKE -16299,0
```

The HGR and HGR2 commands erase the first and second high-resolution pages in memory before doing anything else, and therefore cannot be used for redisplay. The command HGR2 can be used without using HGR first. The commands

`POKE 240,32` and `POKE 240,64`

make HPLOT commands act on page 1 and page 2, respectively, of high-resolution graphics without erasing these first, and regardless of the current display mode.

The command

```
TEXT
```

restores the screen to the text mode. When this is in effect by default or by executing the TEXT command, the display is stored as the first text page in memory. This is preserved when an HGR command switches out of the TEXT mode, and is redisplayed by

```
POKE -16303,0
```

There is a second low-resolution graphics page, and a second text page, that can be used for special applications. They are discussed with other details concerning graphics in reference [3], pp. 131–134.

The construction of shape tables and the commands DRAW, XDRAW, ROT, and SCALE have been described in detail in Sections 3.5 and 3.6.

You may need to override the default boundaries of the region of memory that a large program uses, to prevent graphics overwriting the variables and vice versa. This is done by the commands

`LOMEM:`low-boundary and `HIMEM:`high-boundary

where low-boundary and high-boundary have suitable values.

8.8 USING THE DISK

Writing and reading text files constitute the simplest ways of storing the output of a program on disk and reusing it. Data that are to be used repeatedly can be typed into a program that simply writes them out as a text file. You can write one or you can use utilities, for example, in the DOS tool kit.

A text file can be opened, both for writing and reading, by a statement of the form

```
PRINT CHR$(4);"OPEN";file-name
```

The CHR$(4) sends a CONTROL D code to the output mechanism, thereby switching it from the monitor to the operating system. The further information to the system is transmitted as a character string. The statement to open the file is followed by

```
PRINT CHR$(4);"command-word";file-name
```

where "command-word" stands for "WRITE", "READ", or "DELETE", to start writing a new file (or rewriting an old one), reading a file, or deleting a file, respectively. The data in the first two cases then is written by PRINT statements or read by INPUT statements. A file is closed by

```
PRINT CHR$(4);"CLOSE";file-name
```

During disk input (or output) the reading (or writing) pointer is advanced *k* carriage return codes beyond its current position by

```
PRINT CHR$(4);"POSITION";file-name;k
```

The disk output and input activities are displayed on the screen by executing

```
CHR$(4); PRINT "MON C I O"
```

before these activities begin. The letters C, I, and O cause the operating system commands, the input, and the output, respectively, to be displayed. These can be included or omitted independently. The NOMON command turns the monitoring off.

Much further information on the use of sequential and random disk files is given in reference [4].

A further use of the disk is to "chain" portions of a program that are too large to be in the high-speed memory together. A program called CHAIN on the DOS System Master disk lets you do this. It must be copied onto the same disk as the programs you want to chain, using the very simple procedure described in reference [4], p. 107. Chaining then is performed by executing

```
PRINT CHR$(4);"BLOAD CHAIN,A520":
 CALL 520"name-of-program"
```

where name-of-program is the name, in the catalog, of the program to be brought in. You may need to protect memory by executing LOMEM and HIMEM commands first.

8.9 SYSTEM AND EDITING COMMANDS

In this chapter, commands have been discussed from the standpoint of executing them in programs, that is, in the deferred mode. When you start a session on the Apple, it is in the immediate mode, prepared to execute commands as soon as these are typed. To create a new program, you switch to the deferred mode by typing a number at the start of a line. You type numbers at the beginning of further lines until you want to execute the program. Then you type RUN.

Most of the commands that have been discussed in the preceding sections can be used in the immediate mode, just by omitting line numbers. The Apple can be used as an extended calculator in this way, and graphics can be drawn in both low- and high-resolution modes. You can superimpose the effect of actions in the immediate mode on the results of a program that has executed, and vice versa.

While the immediate mode is the more obvious context in which to introduce the RUN and certain further commands, these can be executed in the deferred mode too. The command RUN, without qualification, restarts the program that is executing. The command

`RUN` line-number

starts or restarts execution at the specified line-number. The command

`RUN` program-name

loads the specified program from disk and executes it. The command

`SAVE` program-name

saves, on disk, under the name that is provided, the program that has just been typed or that is executing.

The command

`LOAD` program-name

loads a new program from disk. This command is essentially for immediate use. The command also can be executed in the course of a program. This gets erased, however, as soon as the new program starts loading.

The command NEW deletes the current program and all its variables. It is essentially for immediate use—to start typing a new program when an earlier one has stopped. It can be used also to make a program self-destruct.

The command

`LIST` starting-line-#, ending-line-#

lists the specified lines. The first and last lines of the program are implied by omitting the first and second operands, respectively. The unqualified command lists the entire program. LIST is used primarily in editing—it can be used during execution, for example, to write part of a program onto disk as a text file. This can be executed subsequently by the EXEC command, to construct a new program that incorporates it.

The command

`DEL` starting-line-#, ending-line-#

deletes the specified lines. It too can be used in both modes. In the deferred mode, execution stops after the deletion and cannot be restarted.

The course and state of action of a program can be interrogated in several ways. In the immediate mode, typing a question mark followed by the name of a variable makes the system display its current value. The amount of free memory is displayed in both modes by

`FRE(`dummy-expression`)`

The current values of LOMEM and HIMEM (the boundaries of the region occupied by the variables) are displayed, respectively, by:

```
PRINT PEEK(116)*256+PEEK(115)
PRINT PEEK(106)*256+PEEK(105)
```

The command TRACE displays the line numbers of statements as these are executed. NOTRACE turns the trace mode off.

There is considerably more detail and opportunity. Enough has been said, however, for present purposes. If you are an expert programmer who branched to this chapter from Chapter 1, branch back to Chapter 2 and use your expertise to full advantage. If you "fell through" from a sequential progression through Chapters 2 to 7, we hope that what you read earlier has helped here, and that this chapter has given you some added insights.

About this book

The senior author is writing these final comments alone, to avoid sentences that get tied in too many knots. This book has been typeset electronically with all the pictures in place by processes that I developed with my wife Barbara in conjunction with Sedgwick Printout Systems on a miniscule budget. Our success is just one more example of the scope for small corporations and people working privately to advance the frontiers of high technology. Looked at from another viewpoint, we have produced a book complete with illustrations entirely from input that we typed ourselves. The commonplace of authors typesetting just their own text may soon be succeeded by many authors typesetting their own text and illustrations in the way we have done.

"Electronic Publishing with Pictures" or EPP for short is a convenient name for the kind of process that we used. Many people are working in this area and typeset pictures have been published previously. For several years, *U.S. News and World Report* has been produced with line drawings and halftone pictures "typeset" in place using commercial equipment that is rather expensive. The second edition of *Interactive Computer Graphics* by Newman and Sproull was produced with pictures in place on an experimental laboratory machine.

Our approach is inexpensive and applies to a broad range of machinery that is used to set type electronically. This expression may seem like a contradiction in terms that is compounded by talking about pictorial typesetting. A short explanation follows.

For centuries metal type was assembled in printing plates that transferred ink to paper for people to read. This simple principle has been elaborated in several ways. One of these is offset printing, commonly used to print books. High quality photographs are made of each page on sheets of light metal. These photographs have special chemical properties. Ink sticks lightly to the areas that were exposed to light, but does not stick to the others. The plates are mounted on rollers and alternately pick up ink and imprint it on paper that is fed through the printing press mechanically.

The originals that are photographed for offset printing can also be produced on electronic typesetting machines such as the Videocomp which we use at Sedgwick Printout Systems. These originals—often called repro copy—are produced photographically too, on a roll of film or paper that passes in front of a very high quality device like a television tube. The text to be printed is displayed line by line or page by page on the screen of this device and focused onto photographic film or paper. This is advanced mechanically after each line or page has been diplayed and recorded.

To drive an electronic typesetting machine, the manuscript is typed into a computer memory together with instructions about the size and style of fonts, the line and page dimensions, and other details of output format. This input is converted to the codes of the electronic typesetting machine. The computer either drives the typesetter directly or it can write the codes onto a magnetic tape that is used to drive the typesetter. Sometimes the codes are punched onto paper tape and sometimes codes are transmitted by satellite to typesetting machines thousands of miles away for rapid publication.

Letters are drawn on the screen by principles that resemble those used in the FONT FORMER program of Section 5.2. Of course the letters are designed on much larger rasters, for example 200 × 200 for the fonts used in this book. Each of the pixels that form the letters is much smaller than the pixels on an Apple monitor. There are nearly fifteen hundred pixels to the inch for the fonts in this book. The fonts are coded in a very compact way for storage in the Videocomp, but the basic idea of constructing a letter from a table of pixel coordinates is the same as in Chapter 5.

Writing the instructions to control the format of the final output is a major item in the preparation of a book for electronic typesetting. Special programming languages called composition languages are used for this. The instructions often are very long and complicated. I designed a major composition language called PAGE-1 when I worked for RCA in 1965. It has been extended to PAGE-2 that is still used widely. We used it to set this book.

My interest in pictorial typesetting started about three years ago when, in my academic work, I was exploring the use of computers to generate printed instructional material. A lot of routine clerical work can be mechanized using word processing techniques, for example, to construct materials that are targeted to different regional groups. Although conventional computer output devices are good enough for much of this, the potential of electronic typesetting merited exploration and I started to look into the possible production of graphs and diagrams on a Videocomp.

I tackled the problem from the standpoint that setting the illustrations would be subsidiary to setting the text, and I experimented with a very simple principle. The position of a character in the output of a Videocomp can be controlled with considerable precision by some of the input commands of the PAGE-2 language. Corresponding provisions exist on other electronic typesetting machines and in other composition languages. Accordingly, I typeset a number of periods in a circular arrangement. By using enough periods to make each one overlap its neighbors I was able to produce very smooth circles. Straight lines were no problem—neither were ellipses, parabolas, sines, exponentials, and other curves that could be defined in simple mathematical terms. These were translated by further programs into streams of codes in the PAGE-2 language to set appropriate patterns of periods. Text that accompanied an illustration was input with just the usual PAGE-2 formatting codes. To check the flexibility of the system I also output some material into two other typsetting languages called MOPAS and TROFF and fed the resulting codes into three systems that accepted these languages, so producing typeset pictures on several models of electronic typesetting machine.

I then started to explore the problem of placing labels interactively on diagrams that had been defined geometrically and ran into a temporary impasse. I was using very sophisticated programming systems on a very large computer and the complexities finally brought my efforts to a halt.

At about this time Graham, who was then fourteen, succeeded in convincing me that the Apple was not a toy by writing prototypes of several programs now in this book. We were forging ahead with some graphics applications of the Apple a little later when a publisher asked me to write a book on graphics. This book is the result. In 1981 the text was typed into an IBM 3033 computer and edited using a system called CMS. In the spring of 1982 it was written onto a magnetic tape that was taken to Sedgwick and loaded onto the UNIVAC 90/60 computer there. It was edited using the UNIVAC text editor program EDT, and a printout sent to the publisher for copy editing. The file in the UNIVAC was updated and the printouts reviewed by the publisher. We got the go-ahead to start typesetting at the beginning of the summer.

Each illustration that depicts the output of an Apple program was produced by creating it on the screen of the Apple and then transmitting to the UNIVAC data that showed which pixels were lighted and which were not. A program in the UNIVAC converted this data to a stream of PAGE-2 codes to typeset the picture. These were injected at the appropriate place in the file of text using simple line editing commands.

Geometrical definitions of the line drawings were typed into the Apple and displayed on the screen. A few programs were used to display and to modify different kinds of drawing interactively. The geometrical definitions were transmitted to the UNIVAC. Further programs on the UNIVAC expanded these to appropriate streams of typesetting codes to create lines and curves consisting of periods that overlapped. These codes were injected into the files of text in the same way as the screen dumps. Still further programs on the UNIVAC allow geometrical definitions of particular kinds of diagram to be scaled and modified in other ways. The placing of labels on pictures has been reduced to simple text editing of the relevant files. So has the movement of pictures in relation to the literal content of the text, and the control of the physical position of pictures on the typeset pages.

Despite all the details of dealing with pictures, the details of dealing with the text to meet conventional demands of book publishing have probably dominated the project. Certainly the tactics to deal with large classes of drawings and diagrams have been codified and stabilized, but the diversity of formatting requirements which publishers will require for the associated text will continue to challenge the most flexible composition languages that can be devised.

The pages have been produced fully made up, cover to cover, without any cutting and pasting. The publisher's logo was dropped onto the title page, and a line in a special font added to the copyright page. The book will go to the printer within a few days of the completion of this epilogue and the indexes.

Many people are working on electronic publishing with pictures and most of what we have done seems fairly obvious, particularly in retrospect. This applies to what went wrong with various false starts that appeared quite reasonable before the event, as well as to the unexpected benefits of the good starts. The amount of detail that has to be considered seems vast in retrospect, but the measures to deal with much of it are now stored on disk and can be taken for granted in the future. Streamlined working procedures now can be established, based on the experience of tasks that led us into the unknown just a few weeks ago.

Subject Index

Absolute value, 175
Aeroplane design, 137
Algorithms. See particular kinds, such as Polynomial evaluation algorithm
Amplitude, 152
Angular speed, 146
Animation effects, 5, 10, 16, 29, 145
Appending data to a file, 33, 181
Applesoft language, 5, 173-184. Also see the individual commands
Appolonius, 61
Arcs, 63
Arc tangent, 175
Arithmetic performed interactively, 12
Armadas, 30
Arrays, 174
Arrows, 16
Arts and crafts, 10
ASCII, 38, 78, 175
Assignment statements, 174
Astroids, 70, 148, 149
Automobile design, 111
Averaging, in suggested variations, 17, 20
Axes, 85, 86, 116, 117, 124, 155
Azimuth, 167

Bar charts, 95-103
Base circle, 69
Bezier fitting, 137
Bezier function, 138, 142
Binary numbers, 14, 48, 49
Binomial probability distribution function, 139
Binomial theorem, 139
Birds, 28
Bricks, 10
Building blocks, 28, 29, 39, 55
Business data, 31
Business graphics, 75
Bypassing faulty input, 34

Captions, 76, 96
Cardioids, 68, 69, 148
Carpet design, 10
Cartesian coordinates, 61
Centipedes, 19
Chaining programs, 88
Charts, 75. See also Bar charts; Pie charts
Chemical formulas, 79
Circles, 57-74
Circloids, 66
Clipping, 122
Coding shape tables, 46, 48, 49
Command languages for graphic output, 23, 24
Communication between computers, 4
Conditional statements, 177
Composition languages, 186-187
Constants, 174
Control points, 141, 143
Contours, 169
Coordinates, 145, 148. See also Cartesian coordinates; Polar coordinates; Screen coordinates; Three-dimensional coordinates
Cosines. See Sines and cosines
Cubes, 22
Cubic curves and equations, 115, 119-121, 130
Cursors, 24, 36-38, 143, 177
Cusps, 69, 148
Cycloids, 111, 148. See also Epicycloids, Hypocycloids

Data structures, 40, 41, 76
Decision making, 75
Defaults, 40
Deferred mode, 182
Deleting text files, 33
Deltoids, 148, 149
Desert islands, 28
Design, 10. See also Interactive design
Diatoms, 50
Difference of squares, 60
Directories in data structures, 40, 41, 79
Disks, 3, 31, 32, 181
Disk drive, 3, 178, 181
Displayed equations, 79, 117, 124, 154, 155
Document transfer, 75
DOS, 173
DOS tool kit, 76
Dot to dot drawings, 138. See also point to point drawings

Editing commands, 182-183
Electrons, 145
Electronic mail, 5, 175
Electronic typesetting, 5, 186-188
Ellipses, 111, 145
Epicycloids, 148. See also Cardioids, Limaçons, Nephroids
Equations. See particular kinds of equations, such as Cubic equations; also Displayed equations
Equations of motion, 112
Error messages, 104
Euclid, 14
Exponential, 175

Families of curves, 68. See also Astroids, Cardioids, Cycloids, Limaçons, Nephroids.
Files. See text files
Floor, 175
Floppy disks. See Disks
Fonts, 76-85, 124, 186
Formulas. See individual kinds, such as Recurrence formulas. See also Equations
Formulas, display of. See Displayed equations
Frame around a graph, 85
French curves, 138
Frequency, 152, 157
Full screen graphics, 54
Functions defined by user, 175
Functions in BASIC, 175

Galileo, 112
Gauss-Jordan elimination, 131
Graphics commands, 179
Graphics mode, 23
Graphics tablet, 4, 75
Graphs, 85-95
Greatest common divisor, 14
Grid on a graph, 85

Halftone pictures, 185
Heart shaped curves. See Cardioids
Heavenly bodies, movement of, 148. See also Planets
Height of trajectory, 112
Hexadecimal numbers, 14, 48, 49
Hidden lines, 169
Hierarchically structured data, 27
High resolution graphics, 53
Histograms, 75
Hulls, 141
Hypocycloids, 148. See also Astroids, Deltoids
Hypotenuse, 62

Immediate mode, 7, 11
Industrial monitoring, 2
Initialization, 174
Input statements, 174
Integers, 174
Interactive design, 31, 36, 50
Isometric projection, 167-168

Jaggies, 58, 59
Joystick, 4

Kaleidoscope effects, 170
Keys to charts, 106
Kidney shaped curves. See Nephroids

Labels on diagrams, 76
Laboratory monitoring, 2
Limaçons, 68
Limits, behavior at, 67

Line numbers, 174

Management control and presentations, 75
Mathematical formulas, display of. See Displayed equations
Matrix inversion, 130-133
Memory, 23
Memory map, 49
Menus, 22, 28, 31, 72, 77, 85, 95, 103, 116, 140, 141, 152, 155, 164
Microprocessor, 3
Modem, 4, 5
Modulus, 176
Monitors, 2

Needlework design, 10
Nephroids, 68,148
Normal equation of a line, 171
Numeric expressions, 175

Orbits, 111, 145

Paddles, 4, 54, 55, 113
Page feature, 52
Page switching, 53, 181
Palm trees, 28
Panoramas, 27
Parabolas, 111, 113, 117, 119
Parametric definition of a curve, 149
Pen plotters, 2
Perspective projection, 163-165
Perspective transformation, 165-166
Phase, 157
Pie charts, 75, 103-110
Pixels, 10, 16, 58, 186-187
Planets, 145
Polar angle, 167
Polar coordinates, 50, 61, 70, 167
Polar equation, 67, 149
Point to point drawing, 22
Polygons, 59
Polynomial evaluation algorithm, 125
Polynomial fitting, 130-133
Polynomials, 111, 113, 115-137
Polyvectors, 180
Positioning commands, 36
Printers, 3, 178
Printing, 186
Processions, 28
Projectiles, 111-114
Prompts, 12, 13, 19, 28, 32, 36, 40, 47, 51, 59, 66, 72, 77, 86, 95-96, 104, 112, 117, 119, 121, 122, 130, 141, 145, 148, 154, 163, 168
Pythagoras, 59, 71

Quadratic curves and equations, 59, 115, 117-119, 130
Quartic curves and equations, 130
Quintic curves and equations, 122, 130

Random numbers, 16, 20, 63, 175
Range of projectile, 112
Rasters, 36, 38-40, 47, 55, 76, 79
Reading text files, 33
Real numbers, 174
Rectification of sine waves, 158
Recurrence formula to evaluate Bezier functions, 139
Repro copy, 186
Rhombus, 66
Robots, 15-18
Roman numerals, 13, 14
Rotation, 78, 79
Rotation equations, three-dimensional, 168
Rotation of shapes, 47, 50, 51, 54
Roulettes, 148. See also Cycloids, Epicycloids, Hypocycloids
Rubber banding, 24

Satellites, 111, 145
Scaling of shapes, 47, 51, 54
Screen color, 54
Screen coordinates, 9, 10
Screen formatting, 79, 177
Secondary menus, 31, 85, 116, 117, 119, 121, 122, 141, 152, 164
Semi-axes, 145
Shading, 88, 125, 157, 169
Shadows, 169
Shapes and shape tables, 46-55
Simultaneous equations, 130-133
Sine waves, 111, 151-161
Sines and cosines, 51, 62, 146, 147, 170, 175
Slide making, 5, 75
Slope intercept equation, 171
Smoothing function in Bezier fitting, 142
Snail shaped curves. See Limaçons
Snowflakes, 50
Speakers, 3, 16, 17, 64, 112, 179
Speech synthesizer, 2
Speed of display, 178
Speed of execution, 174
Spin, 145
Square root, 175
Star gazing, 167
Statements, 174
Storage. See memory
Strings, 174, 175
Subscripts in fonts, 79
Subscripts of arrays, 174
Superscripts, 79
Symmetry, 50, 52, 67, 170
System commands, 33

Tape cassette, 3, 178
Teleconferencing, 5, 75
Text files, 31-33, 43, 76, 79
Text mode, 23, 53
Text window, 180
Texture of lines, 88
Three-dimensional coordinates, 163
Three-dimensional effects, 97, 107
Tick marks, 85, 96, 116, 117, 124, 155
Tracing, 184
Trajectories, 112
Transparencies, 75
Trigonometry, 50, 61, 112, 158. See also Sines and cosines
Turkeys, 23, 24

Unravelled shapes, 48

Variables, 174
Vectors, 180
Videocomp, 186-187
Videodisk, 2

Woodwork design, 10
Writing text files, 32, 33

Keyword Index

Page numbers in italics refer to text discussions. Page numbers in roman type refer to selected uses of each keyword in programs.

ABS, 43, 67, 90,127, 170, *175*
AND, 18, 25, 144, 159, 167
APPEND, *34*, 35
ASC, 38, 81, 95, 102, 109, *175*
AT. See HLIN, VLIN
ATN, *175*

BLOAD, 89, 91, 92, *182*

CALL, 25, *54*, *176*
CALL −868, *179*
CALL −958, *79*, 82, 100, 101, 108, 109, 114, 126, 158, *179*
CALL 520, 91, 92, *182*
CALL 62454, *54*, 64, 65
CHAIN, *88*, 89, 91, 92, *182*
CHR$, *175*
CHR$(4), *33*, 34, 80, 89, 91, 92, 102, 110, *182*
CHR$(7), 15, 90
CHR$(27), *179*
CLEAR, 25, 38
CLOSE, *33*, 35, 36, 81, 89, 102, 110, *182*
COLOR, *8*, 11, 12, *13*, 14, 15, 170, 171, *179*
COS, 64, 65, 73, 108, 109, 114, 151, 169, 170, *175*

DATA, *13*, 14, *23*, 25, 34, 50, 52, *53*, 54, 55, 64, 65, 73, *124*, *174*, *175*
DEF ... FN, *175*
DEL, *183*
DELETE, 35, *182*
DIM, 34, 38, 43, 52, 64, 65, 73, 80, 89, *174*
DRAW, *47*, 50, 54, *181*

END, 25, 31, 36, 45, 144
EXEC, *183*
EXP, *175*

FLASH, 21, 65, 91, 92, 102, 134, 135, 136, 166, 167, *177*
FN. See DEF
FOR ... TO ..., *10*, 123, 14, 15, 18, 21, 22, 25, 34-36, *176*
FOR ... TO ... STEP, 14, 15, 18, 25, 38, 50, 64, 73, 93, 94, *176*
FRE, *183*

GET, 38, 43, 45, 50, 68, 74, 83, 93, 94, 100, *178*
GOSUB, 14, 15, *17*, 18, 21, 38, 39, 44, 73, *176*
GO TO, GOTO, *11*, 12, 14, 15, 18, 21, 22, 25, 30, 34, 36, *176*
GR, *8*, 11, 14, *53*, 170, *179*

HCOLOR, *16*, *17*, 18, 21, 24, 30, 34, 36, 43, 44, 50, 52, *54*, *180*
HGR, *16*, 18, 21, *23*, 24, 30, 34, 36, 38, 39, *53*, 54, *54*, 55, *88*, *180*, *181*
HGR2, *52*, *53*, 54, 88, *88*, *181*
HIMEM, 88, *181*, *182*, *184*
HLIN, *8*, 11, 14, 15, *179*
HOME, *13*, 14, 15, 18, 21, 24, 25, 30, 34, 43, 44, *177*
HPLOT, 18, 21, 24, 30, 31, 34-36, 38-39, *54*, *76*, *180*
HTAB, *13*, 14, 15, 21, 25, 30, 34, 65, 73, 89, 92, *177*, *179*

IF ... THEN, 14, 15, 18, 21, 25, 35, 36, 38, 39, 43, *177*
INPUT, 14, 21, 25, 30, 34-36, 38, 43, 45, *178*
INT, 14, *16*, 18, 64, 65, 66, 81, 108, 114, *175*
INVERSE, 73, 89, 90, 92, *177*
IN#, *178*

LEFT$, *13*, 14, 35, 74, 90, 91, 92, 93, 94, 101, 102, *175*
LEN, 81, 92, 93, 94, 100, 101, 102, 109, *175*
LET, *174*
LIST, *183*
LOAD, *183*
LOG, *175*
LOMEM, 24, 43, 80, 100, 108, 133, 147, *181*, *182*, *184*

MID$, *13*, 81, 92, 93, 94, 95, 127, 129, 135, 136, 160, *175*
MON, *182*

NEW, *183*
NEXT, *10*, 12, 14, 15, 18, 21, 22, 25, 34-36, *176*
NEXT with 2 indexes, 15, 45
NOMON, *182*
NORMAL, 22, 65, 73, 90, 92, 102, 134, 135, 136, 166, *177*
NOTRACE, *184*

ON ... GOSUB, 30, 34, 36, 65, *176*
ON ... GOTO, 30, 34, 43, 45, 73, 80, 81, 82, 83, 89, *176*
ONERR GOTO, 24, 34, 36, *176*
OPEN, *33*, 35, 36, 81, 89, 102, 110, *182*
OR, 25, 91, 92, 160

PDL, *54*, 54, *179*
PEEK, *17*, 18, *65*, 66, *176*
PEEK(−16336), *17*, 18, 66, *179*
PEEK(36), *177*
PEEK(37), 89, 92, *177*
PEEK(105), *184*
PEEK(115), *184*
PEEK(218), *176*
PEEK(222), *176*
PEEK(16384), *178*
PLOT, *9*, 11, 12, 14, 15, 171, *179*
POKE, *24*, 25, 50, 52, *53*, 54, 55, 147, *176*
POKE −16368,0, *178*
POKE −16304,0, *24*, 25, 39, 45, 74, 92, 144
POKE −16303,0, *181*
POKE −16302,0, 52, *54*, 92, *180*
POKE −16301,0, 52, *180*
POKE −16300,0, *52*, 54, *181*
POKE −16299,0, 54, 74, 92, *181*
POKE −16297,0, *24*, 25, 39, 44, *53*, 92, 144
POKE 32,n, *179*
POKE 33,n, *179*
POKE 34,n, 45, *79*, 81, 90, 133, 134, 136, 143, 144
POKE 35,n, *179*
POKE 216,n, 36
POKE 232,n, 50, 52, 54, 55, 147
POKE 233,n, 50, 52, 54, 55, 147
POP, *176*
POS, *177*
POSITION, *182*
PRINT, 14, 15, 21, 22, 25, 30, 34-36, 39, *177*
PR#, *177*, *178*

READ, *13*, 14, 25, 34, 50, 52, 54, 55, 64, 65, 73, *174*, *175*
RECALL, *178*
REM, 11,12, 14, 15, 18, 20, 21, 24, 25, 30, *124*
RESTORE, 15, 25, 73, 80, *175*
RESUME, 25, *176*
RETURN, 14, 15, 18, 21, 30, 31, 34, 35, 44, 45, *176*
RIGHT$, *13*, *175*
RND, *16*, *17*, 18, 64, 65, 66, *175*
ROT, *47*, 50, 52, *53*, 54, *54*, 55, 147, *181*
RUN, *11*, *182*, *183*

SAVE, *183*
SCALE, *47*, 50, 52, *53*, 54, *54*, 55, 147, *181*
SCRN, 171, *180*
SGN, *175*
SIN, 64, 65, 73, 108, 109, 114, 151, 159, 169, 170, *175*
SPC, 108, 167, *177*

SPEED, *178*
SQR, *175*
STORE, *178*
STR$, 92, 93, 101, 109, *124*, 127, 128, 133, 134, 135, 159, *175*

TAB, 108, *177*
TAN, *175*
TEXT, 21, *23*, 25, 34, 38, 39, 43, 44, 45, *53*, 73, 80, *181*
TRACE, *184*

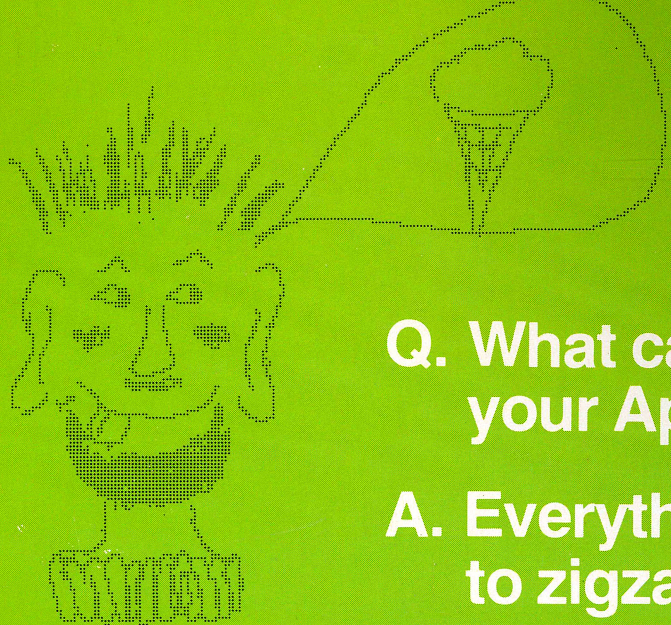
VAL, *13*, 127, 135, *175*
VLIN, *8*, 11, 14, *179*
VTAB, *13*, 14, 21, 25, 30, 34, 44, 45, 52, 60, *177*, *179*

WAIT, *178*
WRITE, *33*, *34*, 35, 81, *181*

XDRAW, *55*, 55, 147, *181*

$14.50

FPT



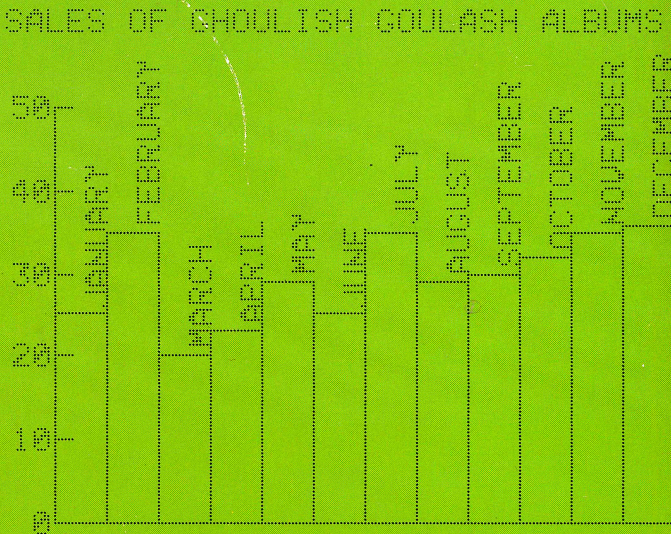
Q. What can you cook up with your Apple® Computer?

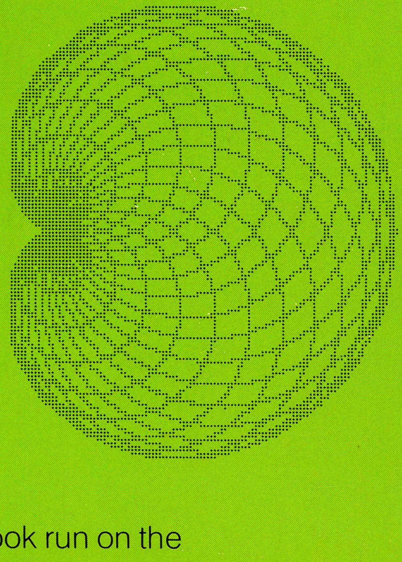
A. Everything from aardvarks to zigzags.

Whether you're a computer professional, a hobbyist, or even a "computer-phobiac," this entertaining and enlightening book will show you just how easy it is to design exciting visual displays in your home, office, or classroom.

Personal Graphics for Profit and Pleasure on the Apple® II Plus Computer teaches you how to create a wide variety of animation effects, perspective projections, geometrics, and much more!

And while *Personal Graphics* is accessible to those without advanced programming and mathematical skills, "computerniks" of all ages and interests will find it to be an excellent source book.





Contents

1 What Is In This Book for You
2 Drawing Some Friends and Foes
3 Building Blocks and Shape Tables
4 Going Around in Circles
5 A Little Plotting and Scheming
6 A Medley of Motley Curves
7 Some Further Directions
8 A Short Account of Applesoft®

Programs in this book run on the
Apple® II Plus computer
with at least 48K RAM
and one disk drive

Little, Brown and Company • Boston – Toronto

01831400

ISBN 0-316-08164-7

PERSONAL GRAPHICS

BARNETT & BARNETT

LITTLE,
BROWN